23
2ej

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

" A R A G O N "

"COMUNICACION CON DISPOSITIVOS EXTERNOS BAJO UNIX A TRAVES DEL PUERTO RS-232"

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A

RICARDO FRANCISCO NUÑEZ ALVAREZ

DIRECTOR DE TESIS: SILVIA VEGA MUYTOY

SAN JUAN DE ARAGON, EDO. DE MEXICO 1993

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

PAG.

CONTENIDO

PAG.

INTRODUCCION

La utilización de las computadoras digitales se ha venido generalizando en el control de procesos, principalmente en los que, por la cantidad de variables que deben manejarse, los sistemas analógicos resultan inadecuados. Los procesos a que se hace mención se encuentran en plantas complejas del tipo petroquímica o bien en las generadoras de energía eléctrica (termoeléctricas, hidroeléctricas, etc). Estas últimas son el punto de partida de las siguientes líneas.

El funcionamiento de una planta generadora es muy complejo, el control de la misma va desde los procesos de abastecimiento de recursos para el funcionamiento de la planta hasta la distribución de la energía eléctrica. Para llevar a cabo dicho control, se están incorporando computadoras digitales a dichos procesos (siguiendo con la tendencia antes mencionada) y llevando la información proveniente de estos procesos hacia lo que se denomina CENTRO DE CONTROL.

Algunas de las funciones principales que deben llevar a cabo estos centros de control son: la obtención y procesamiento de la información, monitoreo de los procesos, análisis estadístico y de seguridad.

Para lograr que un centro de control realice satisfactoriamente estas funciones, es necesario tener una configuración adecuada del mismo. Existen diferentes configuraciones de estos centros de control, entre las cuales tenemos el sistema dual de computadoras en que, una de ellas lleva el control mientras que la segunda se encuentra en espera de actuar cuando la primera falle; otra configuración es separar las funciones de *adquisición* de las funciones de *procesamiento* de información (almacenamiento y control). Bajo el principio de esta última se encuentran en la actualidad los sistemas SCADA que llevan a cabo dichas funciones de adquisición, procesamiento y control.

Estos sistemas SCADA o sistemas de control supervisorio y adquisición de datos (Supervisory Control And Data Acquisition), están siendo aplicados ampliamente en el extranjero a plantas de generación de energía.

Para llevar a cabo la función de adquisición de información, los sistemas SCADA se auxilian de dispositivos denominados UTR (Unidades Terminales Remotas), que adquieren información digital y/o analógica y la envían a un dispositivo de almacenamiento y control de información vía un canal de comunicaciones (línea telefónica por mencionar alguno). Este dispositivo de almacenamiento y control, al cual llega la información proveniente de las UTR, es denominado Estación Maestra dentro de la terminología de los sistemas SCADA.

En el presente trabajo de tesis, se desarrolló el software para el sistema de comunicaciones de una PC que trabaja como estación maestra en un sistema piloto de supervisión y control basado en la arquitectura SCADA.

Este sistema piloto, emplea una PC como dispositivo para el almacenamiento y control, la cual se comunica con el dispositivo adquisitor mediante un puerto serie RS-232. Dicha PC opera con el sistema operativo VENIX (que es una implementación del UNIX V5R4), de la que se deriva el título de este trabajo:

COMUNICACION CON DISPOSITIVOS EXTERNOS BAJO UNIX A TRAVES DEL PUERTO RS-232

Este software encargado del manejo del sistema de comunicaciones para la transferencia de información entre la PC (con sistema operativo UNIX) y el dispositivo adquisistor (dispositivo externo) cumple con las siguientes funciones:

- Selección de los parámetros del puerto.
- Recepción de información.
- Transmisión de información.
- Verificación del estado del puerto.

Para poder implementar estas funciones fue necesario primeramente conocer el puerto serie RS-232 y posteriormente aplicar una técnica de programación que nos llevara a obtener un software terminal que cumpla con dichas funciones. Por ello, la organización de la tesis es la siguiente:

El objetivo del capítulo I es proporcionar la teoría necesaria sobre el puerto serie. Para ello, se revisan primeramente los términos que se usarán a través del capítulo, continuando con una introducción a uno de los dispositivos que manejan comunicación serie asíncrona: el UART; como siguiente punto, se tiene la revisión sobre algunos puntos del estándar RS-232, finalizando el capítulo con el UART 8250.

Contando con los conocimientos básicos, en el capítulo II el objetivo es desarrollar el software que cumplirá con las funciones de comunicación. Para lograr el desarollo de dicho software se usa el ciclo de vida clásico de software que consta

de las siguientes etapas: 1° Requerimientos del sistema, 2° Análisis de requerimientos, 3° Metodología de diseño, 4° Codificación, 5° Pruebas y 6° Mantenimiento. De este ciclo de vida clásico, el capítulo sólo cubre los cuatro primeros puntos.

El capítulo III cubre la quinta etapa del ciclo clásico para el desarrollo del software: Pruebas. Este capítulo consiste de tres puntos: el primero toca lo referente a las pruebas que se hicieron entre dos PC, ambas con sistema operativo DOS; el segundo punto trata de las pruebas realizadas también con dos PC, pero en este caso una con sistema operativo DOS y la otra con UNIX; esta última con el software desarrollado en el capítulo II y el tercer punto son las pruebas realizadas con un dispositivo externo con la condicional de no ser éste una PC.

Para finalizar con la introducción, sólo resta mencionar que el resultado del punto 4 del ciclo de vida clásico, que corresponde a la codificación, se presenta en el apéndice A, mostrado como la parte final de todo el trabajo, mientras que la 6ª etapa se realizará posteriormente a medida que el software sea utilizado y depurado, en caso necesario, para cumplir con nuevos requerimientos o subsanar posibles errores.

CAPITULO I

ESTANDAR RS-232.

INTRODUCCION:

Existen básicamente dos tipos de comunicación serie: asíncrona y síncrona. La comunicación asíncrona es empleada, hoy día, en numerosas aplicaciones (como en el caso de las PC) y para su implementación se hace uso de normas (como es la RS-232).

El objetivo del capítulo es el estudio de la comunicación serie asíncrona. Para cubrir dicho objetivo es necesario definir términos comúnmente usados en este tipo de comunicación. Una vez definidos dichos términos se hará un estudio de uno de los dispositivos que permite llevar a cabo la comunicación serie asíncrona: el UART.

1.1 TERMINOLOGIA EN COMUNICACION SERIE ASINCRONA UTILIZANDO EL ESTANDAR RS-232.

La comunicación serie asíncrona tiene una extensa gama de términos así que sólo se cubrirán los de interés para el estándard RS-232.

A).- El primer término a definir es *comunicación asíncrona*". De manera estricta, un evento asíncrono es aquel que se genera en un instante aleatorio o no determinístico. Para el caso de la *comunicación asíncrona*, ésta se tiene si los elementos que la integran intercambian información en tiempos aleatorios o en instantes no conocidos.

B).- El *canal de comunicaciones*" es el medio entre el transmisor y el receptor por el cual las señales (eléctricas, ópticas, etc.) viajan.

C).- El *bit de inicio*" es enviado por el transmisor para indicar al receptor que comienza la información. Tiene como objetivo el sincronizar el transmisor con el receptor poniendo a este último en espera de información.

D).- Los *bits de dato*" es la información principal a intercambiar durante la comunicación. A pesar de que en el canal de comunicaciones viajan los bits de inicio, los bits de dato y otra información, los bits de dato es la información que finalmente se quiere transferir.

E).- En algunos casos el canal de comunicaciones está expuesto a inducción de ruido de alguna fuente. Para evitar que se reciba información errónea, el transmisor envía el *bit de paridad*" que le permite al receptor verificar que los bits de dato fueron recibidos correctamente. Este bit de paridad es un bit de detección de errores.

F).- El *bit de paro*" es enviado por el transmisor para indicar a su contraparte el fin de la transferencia de información o bien de la comunicación que en ese momento se encontraba establecida.

G).- Una *trama* es el total de: bit de inicio, bits de dato, bit de paridad y bit de paro, que es enviada por el transmisor tal como se muestra en la figura I.1.

H).- Un *modem*[4] es un dispositivo que permite *mod*ular una señal y enviarla por el canal de comunicaciones, o bien *dem*odular la señal que le llega del canal de comunicaciones. En el estándar, el modem corresponde a un DCE, mismo que se define más adelante.

I).- El término *bauds*[b,c] es utilizado en telefonía. Este término y el de *bits por segundo* son, a menudo, usados indistintamente; pero existe, en algunos casos, una diferencia entre dichos términos. Para entender el término de baud es necesario definir primeramente lo que es un *símbolo*.

El término de *símbolo*, en su forma estricta, se define como la representación de algo único. En equipos de transmisión, la palabra *símbolo* se usa para referirse a una forma de señal específica; por ejemplo: una señal senoidal con amplitud, frecuencia y fase específica es un *símbolo*. Al cambiar alguna de sus carcterísticas (la fase, por ejemplo) entonces se tendrá un *símbolo* diferente; otro ejemplo son las señales multinivel (M-aria), en la que cada nivel corresponde a un *símbolo*.

En las líneas telefónicas se habla de transmisión de *símbolos/segundo* como la razón entre el número de *símbolos* transmitidos en un segundo. En el caso de los modems, y de algunos otros equipos, la unidad de medida para el número de símbolos transmitidos en un segundo sobre la línea telefónica es el *Baud*.

Los *bits por segundo (bps)* son la medida del número de bits (símbolos binarios) por segundo. Cuando los símbolo en el canal de comunicaciones corresponden a señales binarias, puede hablarse de *bps*, *símbolos/segundo* o bien *Bauds*.

Cuando dos o más bits son empaquetados en un *símbolo* para que éste sea enviado por el canal de comunicaciones, entonces los *bps* no corresponden a *Bauds* o *símbolos/segundo*, aunque el número de símbolos sea dos.

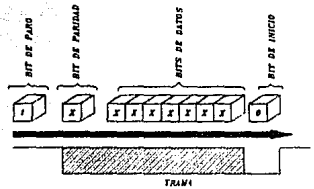

Figura I.1 Trama con bits de inicio y paro.

En el caso del modem, comúnmente los bits son empaquetados y modulados para ser enviados por el canal de comunicaciones, haciendo que los *bps* y los *bauds* no correspondan.

Cuando en una PC se tiene una configuración de modem nulo (mostrada más adelante) para la comunicación, entonces se puede hablar indistintamente de bps o bien de bauds.

J).- Un *DTE*[8] es un equipo terminal de datos. Este equipo es la fuente o destino de la información; esto es, el dispositivo a donde se quiere que la información llegue o de donde la información se obtiene.

K).- La parte complementaria del DTE es el *DCE*. Un *DCE* es un equipo de comunicación de datos, y como su nombre lo indica, es aquel que se encarga de hacer el enlace entre DTE's (locales o remotos) vía un canal de comunicaciones. Este proceso se explica más a detalle en secciones posteriores.

L).- Los términos *MARCA* y *ESPACIO* son el equivalente a un 1 y un 0 lógico o niveles de alto y bajo respectivamente. Estos términos tienen sus orígenes en la telegrafía.

M).- Por último, las siglas *EIA* y *RS* utilizadas en la nomenclatura del estándar corresponden a Electronic Industries Association y a Recommended Standard respectivamente.

I.2 EL UART.

El *UART* (Universal Asynchronous Receiver-Transmitter) es el dispositivo (comúnmente un CI) mediante el cual se lleva a cabo la comunicación asíncrona.

Para llevar a cabo esta comunicación, el UART sigue un proceso que se presenta en las figuras I.2 y I.3. La figura I.2 corresponde al diagrama de flujo para una transmisión, mientras en la figura I.3 se muestra el diagrama de flujo para una recepción. Los diagramas manejan: un bit de inicio, 8 bits de datos, un posible bit de paridad y un bit de paro; sin embargo, pueden manejar otros parámetros.

Para la transmisión, la secuencia es:

1.- BIT DE INICIO: Se pone la salida en ESPACIO (OFF o cero lógico). Cada bit permanece en la salida hasta la llegada de otro durante un cierto tiempo calculado en base a la velocidad de comunicación.

2.- ENVIO: Se envía bit por bit, de los datos, partiendo con el bit menos significativo.

3.- BITS DE PARIDAD: Una vez enviados todos los bits de información, el siguiente paso es la paridad. En el caso de ser requerida, deberá de calcularse con anterioridad[1].

[1] *En algunos casos el bit de paridad es introducido entre el mensaje y no al final del mismo como ocurre aquí.*

4.- BITS DE PARO: Para culminar con la transmisión, se envía el bit de paro. Se pone la salida en Marca (ON o uno lógico) y se mantiene así todo el tiempo hasta una nueva transmisión.

Para la recepción el proceso es:

1.- BIT DE INICIO: Se monitorea el canal de comunicación hasta identificar el bit de inicio enviado por el transmisor.

2.- BITS DE DATO: Una vez detectado éste, se hace una espera hasta recibir el primer bit de información y en ese instante se realiza la lectura de los bits de entrada. El proceso de lectura bit a bit involucra un desplazamiento continuo hasta completar todos los bits del dato a recibir.

3.- BITS DE PARIDAD: Después de haber recibido todos los datos, el siguiente paso es verificar la existencia de error en la transmisión (si así se requiere) mediante el bit de paridad. En caso de error, éste se reporta.

4.- BITS DE PARO: Para terminar la recepción se debe leer el bit de paro indicando así el fin de la comunicación.

Un UART con funciones básicas podría tener el diagrama de la figura I.4

Cuando a este UART se le implementan elementos para el manejo de algún estándar resulta un CI de gran utilidad y fácil manejo como lo son los UART 8250[4] y el 16450 utilizados con las PC's y cuyo diagrama a bloques se muestra en la figura I.5.

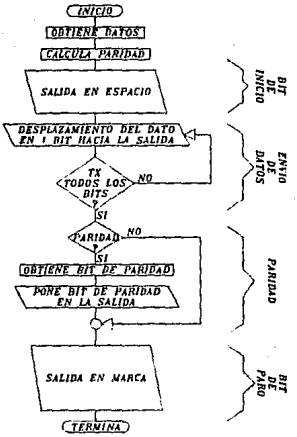

Figura I.2 Diagrama de flujo del transmisor.

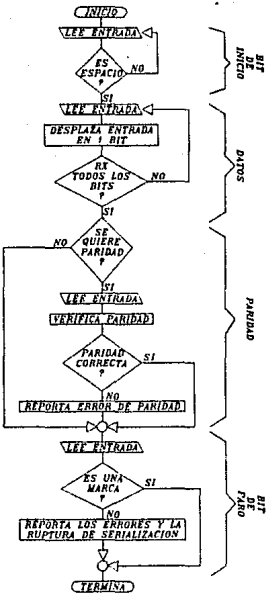

Figura I.3 Diagrama de flujo del receptor.

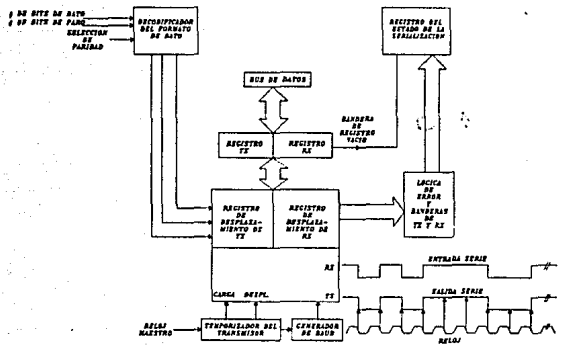

Figura I.4 UART a bloques.

I.3 ESTANDAR RS-232.

El nombre formal del RS-232 es " INTERFAZ ENTRE EQUIPO TERMINAL DE DATOS Y EQUIPO DE COMUNICACION DE DATOS EMPLEANDO INTERCAMBIO SERIAL DE DATOS BINARIOS "[2].

El estándar RS-232 es un documento que especifica una norma recomendada (Recommended Standard) para la comunicación entre un dispositivo DTE y un dispositivo DCE". El DTE es un dispositivo que requiere enviar o recibir información a otro dispositivo, haciendo uso de algún canal de comunicaciones (comúnmente la línea

[2] *Interface between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange.*

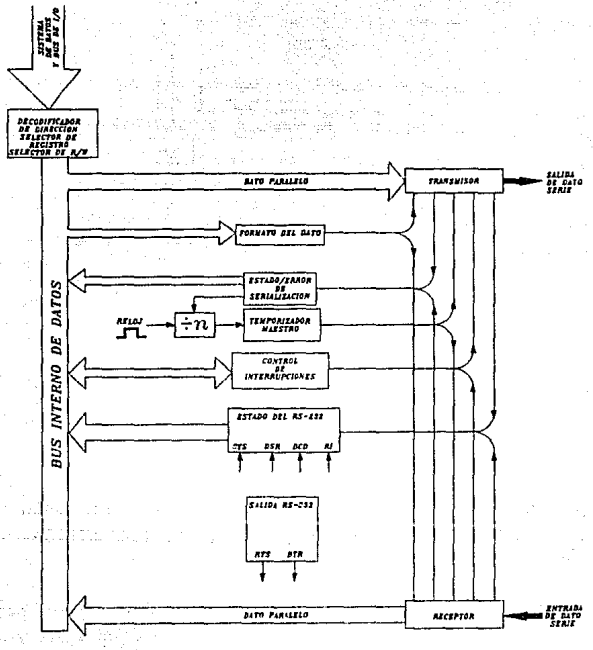

Figura I.5 UART 8250.

telefónica). El DCE es un dispositivo que servirá como intermediario, por así decirlo, para que el DTE haga uso del canal de comunicaciones.

I.3.1 ESPECIFICACIONES DEL ESTANDAR RS-232.

CONECTOR EN EL RS-232: El estándar define un conector de 25 pines denominado DB-25. Este conector, de acuerdo al estándar, *puede* ser un conector macho para el DTE y un hembra para el DCE aunque no necesariamente.

PINES Y SEÑALES EN EL CONECTOR: Cada una de las líneas o pines en el conector tiene definida, una señal tal como se muestra en la tabla I.1.

TABLA I.1: Asignación de señales para los 25 pines.

PIN	SEÑAL	DESCRIPCION
1	AA	TIERRA DE PROTECCION.
2	BA	DATO TRANSMITIDO (TD).
3	BB	DATO RECIBIDO (RD).
4	CA	REQUERIMIENTO PARA ENVIAR (RTS).
5	CB	LISTO PARA ENVIO (CTS).

PIN	SEÑAL	DESCRIPCION
6	CC	DCE LISTO.
7	AB	COMUN.
8	CF	DETECTOR DE SEÑAL DE LINEA RECIBIDO (RLSD).
9	--	RESERVADA.
10	--	RESERVADA.
11		SIN ASIGNACION.
12	SCF/CI	RLSD SECUNDARIO.
13	SCB	CTS SECUNDARIO.
14	SBA	TD SECUNDARIO.
15	DB	TEMPORIZADOR DEL DCE.
16	SBB	RD SECUNDARIO.
17	DD	TEMPORIZADOR DEL
18	LL	DCE.
19	SCA	AUTOPRUEBA LOCAL.
20	CD	RTS SECUNDARIO. DTE LISTO.
21	RL/CG	AUTOPRUEBA
22	CE	REMOTA.
23	CH/CI	INDICADOR DE TIMBRE.
24	DA	SEÑAL DE VELOCIDAD.
25	TM	TEMPORIZADOR DEL DTE. MODO DE PRUEBA.

Es posible tener una aplicación que no requiera utilizar el total de las señales, pudiéndose eliminar su conexión o simplemente no tomarse en cuenta. IBM en su implementación del estándar en la PC consideró únicamente 9º, mismas que se enlistan en la tabla I.2.

TABLA I.2: SEÑALES DEL RS-232 PARA I/O ASINCRONA EN PC.

PIN	ABREV.	NOMBRE	DIRECCION	FUNCION
2	TD	DATO TRANSMITIDO	HACIA EL DCE	DATO A TRANSMITIR POR EL DTE
3	RD	DATO RECIBIDO	HACIA EL DTE	DATO A RECIBIR POR EL DTE
4	RTS	REQUERIMIENTO DE ENVIO	HACIA EL DCE	EL DTE QUIERE TRANSMITIR
5	CTS	LISTO PARA ENVIAR	HACIA EL DTE	EL DCE ESTA LISTO PARA ENVIAR LO DEL DTE
6	DSR	DCE LISTO	HACIA EL DTE	DCE LISTO PARA COMUNICARSE

PIN	ABREV.	NOMBRE	DIRECCION	FUNCION
7	---	SEÑAL DE COMUN	---	LINEA DE COMUN PARA LA CIRCUITERIA
8	DCD	DETECTOR DE PORTADORA (RLSD)	HACIA EL DTE	ENLACE EN PROCESO
20	DTR	DTE LISTO	HACIA EL DCE	DTE LISTO PARA COMUNICARSE
22	RI	INDICADOR DE TIMBRE	HACIA EL DTE	ANUNCIA LA ENTRADA DE UNA LLAMADA

Una descripción funcional de las señales utilizadas en la implementación a la PC de IBM es la siguiente:

DATO TRANSMITIDO (Señal: TD "Transmitted Data", pin 2).
Dirección: Del DTE.

Esta señal es generada por el DTE transfiriendo el dato al DCE. De acuerdo con la convención ya establecida, el DTE mantiene esta señal en condición de MARCA (1 lógico) durante estado de no transmisión.

El DTE no podrá transmitir datos si no existe una condición de MARCA en las siguientes señales:

1.- Requerimiento de envío, RTS (señal del pin 4).
2.- Limpio para envío, CTS (señal del pin 5).
3.- DCE listo, DSR (señal del pin 6).
4.- DTE listo, DTR (señal del pin 20.)

DATO RECIBIDO (Señal: RD "Received Data", pin 3).
Dirección: Al DTE.

La línea de dato recibido lleva el dato serie del DCE al DTE. Esta señal es enviada por otro DCE remoto o local. Esta señal se mantiene en condición de MARCA durante estado ocioso.

REQUERIMIENTO DE ENVIO (Señal: RTS "Request To Send", pin 4).
Dirección: Del DTE.

Esta señal es una condición para indicar al DCE que el DTE quiere transmitir. Esta señal se encuentra en ESPACIO (OFF o 0 lógico) durante estado ocioso.

LISTO PARA ENVIO (Señal: CTS "Clear To Send", pin 5).
Direccion: Al DTE.

Esta señal es generada por el DCE para indicarle al DTE que su petición de transmitir información puede llevarse acabo. Esta señal es la respuesta del DCE cuando el DTE realiza un requerimiento de envío mediante la señal RTS (pin 4). Cuando esta señal se encuentra en MARCA el DTE podrá transmitir y cuando se encuentra en ESPACIO el DTE no podrá transmitir.

DCE LISTO (Señal: DSR "Data Set Ready", pin 6).
Dirección: Al DTE.

Esta señal es usada para indicar el estado del DCE. Esta señal es enviada al DTE informándole que el DCE se encuentra conectado. La condición de MARCA en esta señal indica que:

1.- El DCE está conectado al canal de comunicaciones.
2.- El DCE no realiza alguna de las funciones con la línea de comunicación.

Una condición de ESPACIO indica que el DCE no se encuentra disponible para la comunicación.

Cabe recalcar que el término DCE fue remplazado por el de " Data Set " y que éste último es el nombre que la compañía telefónica norteamericana da a un modem.

COMUN DE LA SEÑAL (Señal: GND "Ground", pin 7).
Dirección: Ambas.

Este es el común o referencia, que se encuentra en todos los circuitos. Esta señal es diferente a la tierra física (pin 1), por lo que no debe existir conexión entre ambas.

DETECTOR DE PORTADORA (Señal: DCD "Data Carrier Detect", pin 8).
Dirección: Al DTE.

Esta señal, cuyo nombre oficial es Detector de Señal Recibida de Línea, es habilitada (puesta en MARCA) cuando el DCE recibe una portadora remota indicando que un DTE remoto o local quiere establecer un enlace. Es mantenida en una MARCA durante el enlace.

TERMINAL DE DATO LISTA (Señal: DTR "Data Terminal Ready", pin 20).
Dirección: Del DTE.

Esta señal tiene dos funciones:

1º Indica al DCE que el DTE se encuentra en operación.
2º Prepara al DCE para que se conecte a la línea telefónica. Esto significa que DTR habilita al DCE para que se haga un enlace con la línea y poder llevar acabo la transmisión o recepción.

Una vez que el DCE es conectado a la línea, DTR debe permanecer habilitada para mantener la conexión, la inhibición causa la desconexión del DCE de la línea de comunicaciones interrumpiendo el enlace de datos en proceso.

INDICADOR DE TIMBRE (Señal: RI "Ring Indicator", pin 22).
Dirección: Al DTE.

Este pin es habilitado durante una llamada en la línea. El RI es habilitado en MARCA aproximadamente un tiempo igual al de un timbrazo en el teléfono y mantenido en ESPACIO el resto del tiempo. Esta señal aparece independiente del estado de Data Terminal Ready (señal 20).

Un aspecto de gran importancia es que los dos extremos de la interfaz RS-232 son *complementarios*, una salida en el DTE es una entrada en el DCE. Otro aspecto de importancia es que el nombre de cada señal es determinado a partir del DTE como referencia, por ejemplo "Dato Transmitido" denota una información de salida del DTE. La tabla I.2 proporciona las direcciones de las señales. Esta identificación de la dirección en las señales es lo que diferencia un DTE y un DCE, como se observa en la figura I.6.

CLASIFICACION DE LAS SEÑALES: Las señales de la RS-232 pueden dividirse en: *señales de tierra, señales de datos, señales de control* y *señales de temporización*. Las *señales de tierra* son básicamente el común o referencia para los niveles de voltaje y la tierra física como protección del sistema a descargas eléctricas o por estática. Las *señales de datos* son simplemente lo transmitido y lo recibido en los pines 2 y 3, respectivamente. Las *señales de control*, nombradas así por llevar acabo el proceso de *HANDSHAKING*[1], son estados o comandos para controlar el proceso de comunicación. Por último las *señales de temporización* son utilizadas para llevar a cabo la sincronización en una comunicación síncrona.

VELOCIDAD DE COMUNICACION: El estándar especifica velocidades desde CERO hasta un límite de 20, 000 bits por segundo. En la mayoría de los sistemas, la velocidad de datos es limitada a 19,200 bps. El estándar también prevee longitudes de cable de hasta 50 pies.

NIVELES DE VOLTAJE: Los niveles lógicos son bipolares. Estos son niveles lógicos que son representados no solamente por su magnitud sino también por su polaridad. Los voltajes máximos permitidos en cualquiera de las señales son ± 15 Volts.

El estándar RS-232 actualmente define *cuatro* niveles lógicos. Los niveles de las señales de entrada - de control y de dato - son diferentes a los niveles de las señales de salida - también de control y de dato -. La figura I.7 muestra estos niveles lógicos que son, para las salidas, de +5 a +15 y de -5 a -15 (los voltajes entre +5 y -5 no se encuentran definidos); para las entradas son de +3 a +15 y de -3 a -15 (los voltajes entre +3 y -3 no se encuentran definidos). La diferencia entre los niveles lógicos de salida a entrada (5 - 3 = 2 V) es referida como el *MARGEN DE RUIDO*, esto significa que la interfaz puede tolerar 2 volts pico de ruido entre éstas.

[1] *HANDSHAKING: El intercambio de señales predeterminado entre dos dispositivos para establecer la conexión entre ellos.*

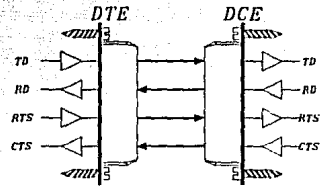

Figura I.6 Sentido de algunas señales entre DTE y DCE.

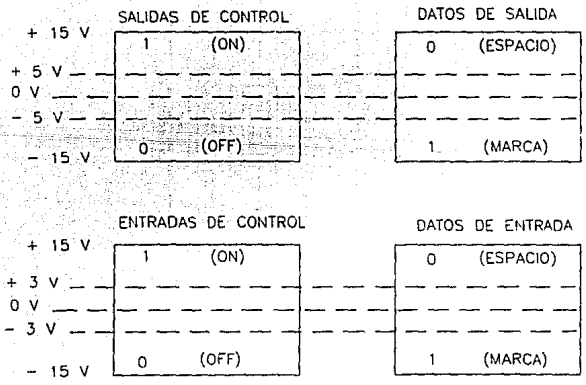

Figura I.7 Niveles de voltaje en el estándar.

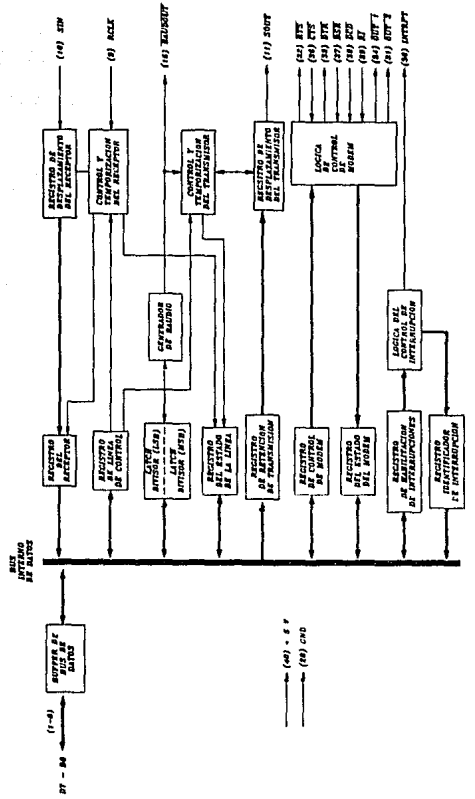

Figura I.8 Diagrama a bloques del UART 8250.

I.4 EL UART 8250.

El diagrama a bloques de la figura I.8 muestra el UART 8250 con sus funciones para el manejo de las señales del estándar.

Este circuito integrado tiene 10 registros que permiten programar los parámetros del puerto (bit de inicio, número de bits de dato, tipo de paridad y número de bits de paro) por medio de software, así como saber el estado del puerto y las señales del estándar.

A continuación se presenta la tabla I.3 con la lista de los 10 registros[1]. Como se puede observar en la tabla I.3, cuando se lleva a cabo una comunicación serie con un modem (o DCE) se puede hacer uso de los registros para monitorear y controlar el estado del modem y del puerto (estado de la línea); pero cuando se tiene una comunicación con modem nulo los registros destinados para el modem podrán pasarse por alto.

TABLA I.3 Registros del 8250 y direcciones en PC

No	NOMBRE DEL REGISTRO	DIRECCION	CONDICIONES
1	TRANSMITTER HOLDING	3F8H ESCRITURA	BIT_7 = 0 EN 3FBH
2	RECEIVER DATA	3F8H LECTURA	BIT_7 = 0 EN 3FBH
3	BAUD-RATE DIVISOR (BYTE BAJO)	3F8H LECTURA/ESCRITURA	BIT_7 = 1 EN 3FBH

No	NOMBRE DEL REGISTRO	DIRECCION	CONDICIONES
4	BAUD-RATE DIVISOR (BYTE ALTO)	3F9H LECTURA/ESCRITURA	BIT_7 = 1 EN 3FBH
5	INTERRUPT ENABLE	3F9H LECTURA/ESCRITURA	BIT_7 = 0 EN 3FBH
6	INTERRUPT IDENTIFICATION	3FAH LECTURA	NO EXISTEN
7	LINE CONTROL	3FBH LECTURA/ESCRITURA	NO EXISTEN
8	MODEM CONTROL	3FCH LECTURA/ESCRITURA	NO EXISTEN
9	LINE STATUS	3FDH LECTURA	NO EXISTEN
10	MODEM STATUS	3FEH LECTURA	NO EXISTEN

Las direcciones[1] que aparecen en la tabla corresponden a las del puerto COM1 en la PC y con ellas se puede accesar a los registros con una simple instrucción de IN y OUT (ensamblador o su equivalente en cualquier lenguaje) para su lectura o escritura.

Debido a que los voltajes y los niveles lógicos de la RS-232 no son compatibles con el 8250, es necesaria una conversión de niveles. Esto se realiza mediante un Circuito Integrado especial conocido como EIA (RS-232) Driver. El 8250 maneja las señales del modem invertidas por lo que el EIA Driver deberá realizar la conversión. La figura I.9 muestra, por ejemplo, como las señales de RTS y el CTS del estándar RS-232 son RTS y CTS negadas en el 8250 debiéndose realizar la conversión e inversión de niveles.

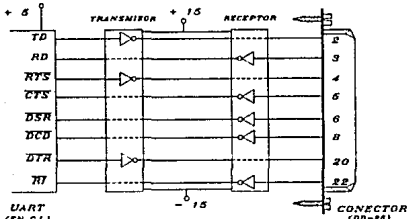

Figura I.9 Conversión TTL a RS-232.

I.5 MODEM NULO.

El modem nulo, comunmente, es una conexión directa de dos computadoras que trabajan como DTE. Para ello es necesario hacer un arreglo con las líneas del estándar de modo que las computadoras (o DTE's) operen como si se encontraran entre ellas los modem (DCE's).

Para llegar a la configuración de modem nulo, considerese primeramente la figura I.10 donde se observa que los pines se conectan directamente entre la computadora (DTE) y el modem (DCE) debido a que sus señales son complementarias. Por esta razón, cuando tenemos dos dispositivos que se encuentran operando como DTE y se requiere una conexión directa entre ambos es necesario realizar un arreglo como el de la figura I.11 donde se muestra la implementación de un modem nulo para transmitir en un sentido.

Existen diversas conexiones de modem nulo, dependiendo de la aplicación que se requiera. Si alguna aplicación no requiere handshaking, no se tomarán en cuenta las señales de control del modem, y podrán conectarse únicamente las líneas de datos y el común o bien, si la aplicación requiere saber la presencia de los dispositivos a comunicar, basta con conectar y sensar la señal de DTR (Data Terminal Ready) además de las señales de dato.

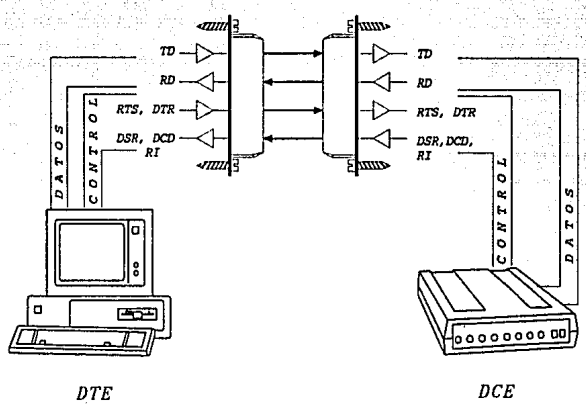

Figura I.10 PC - MODEM como DTE - DCE respectivamente.

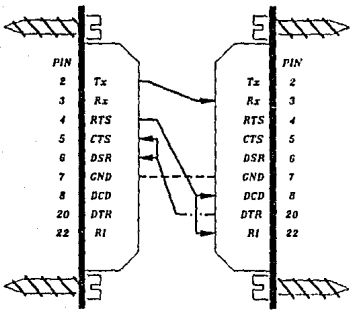

Figura I.11 Conexión modem nulo entre PC's.

CAPITULO II

DESARROLLO DEL SOFTWARE

INTRODUCCION

Para desarrollar el software que permitirá tener un control sobre el sistema de comunicaciones, fue necesario el conocimiento adquirido en el capítulo I para con ello, y junto con un proceso de desarrollo, se pueda llegar al software que cumpla con las especificaciones u objetivos planteados.

Del capítulo I resalta, como un punto importante, el CI 8250 como un manejador para comunicación serie asíncrona. Este CI, como un recurso, se encuentra en la mayoría de las implementaciones del puerto serie asíncrono en las PC. Este CI permite:

- Transmisión y recepción serie asíncrona.

- Elección de los parámetros de configuración para una transmisión asíncrona como son: Bauds, bits de dato, tipo de paridad y bits de paro.

- Verificación de la condición del puerto como son : Detección de errores por una paridad incorrecta o bien que el canal de comunicaciones se encuentre abierto al recibir una trama de ceros.

La característica primordial de este CI es que su configuración y control es vía software. Por ello, el objetivo del presente capítulo es desarrollar el software para controlar el puerto serie (a través del 8250). Este desarrollo de software implica la especificación y el análisis de los requerimientos que llevará éste, así como una metodología a seguir. El proceso de desarrollo de software aplicado es conocido bajo el nombre de: ciclo de vida de software[1].

II.1 ETAPAS EN EL DESARROLLO DEL SOFTWARE.

El proceso para el desarrollo de software tiene la siguiente secuencia[h]:

1º Definición de requerimientos.
2º Análisis de requerimientos.
3º Elección de la metodología de diseño.
4º Codificación.
5º Pruebas.
6º Mantenimiento.

Los cuatro primeros puntos serán tratados en el presente capítulo y el quinto es tratado en el capítulo siguiente, mientras que el último se aplicará cuando se realicen actualizaciones o correcciones al software conforme surjan nuevos requerimientos.

[1] *Algunos autores se refieren a él como "ciclo de vida clásico".*

II.2 PRIMERA ETAPA: REQUERIMIENTOS DEL SISTEMA.

La **definición de los requerimientos** es la especificación de los puntos funcionales que deberá satisfacer el software a desarrollar y si es necesario, se harán especificaciones de hardware, recursos humanos, técnicos, etc.

Los requerimientos del software para el control del puerto serie son:

A).- Realizar una autoprueba al CI 8250 para verificar su operabilidad.
B).- Habilitar el puerto de comunicaciones con parámetros inicialmente conocidos.
C).- Permitir la configuración de los parámetros del puerto de comunicaciones (Bauds, bits de dato, tipo de paridad y bits de paro).
D).- Permitir la transmisión y recepción de información mediante peticiones.
E).- Permitir la recepción de información en cualquier momento sin que se haga una petición (recepción asíncrona).
F).- Permitir la verificación del estado del puerto en cualquier instante.
G).- Permitir al usuario interactuar con el sistema mediante ventanas y menúes gráficos.

A manera de abreviación, y en lo posterior, al software para el control del puerto de comunicaciones asíncrono se referirá como **módulo de comunicaciones.**

II.3 SEGUNDA ETAPA: ANALISIS DE REQUERIMIENTOS.

El **análisis de requerimientos** consiste en detallar la función que el software debe cumplir a partir de los requerimientos impuestos; es decir, se describe el "qué" debe hacer y "cómo" debe lograr su propósito, apegándose a los requerimientos de software, hardware, humano, técnicos, etc.

Para realizar el análisis de los requerimientos se examina cada uno de los mismos por separado para determinar "qué" hará. Una vez determinada la función de cada uno, se resuelve el "cómo" lo hará; para ello se desglosa cada requerimiento en niveles de abstracción que permitan llegar a la solución. El resultado final de este análisis de requerimientos es la solución al sistema presentado como un conjunto de subsistemas fáciles de entender.

II.3.1 EL "QUE".

Por lo anterior, la primera parte en el análisis de requerimientos define el "qué" debe hacer el software:

A).- Realizar una autoprueba al CI 8250 para verificar su operabilidad.

DESCRIPCION : Detectar la existencia física del puerto de comunicaciones mediante una autoprueba realizada al CI 8250. Esta autoprueba a su vez permite verificar la correcta operación de dicho CI. El resultado de la autoprueba deberá ser reportado al usuario en una ventana gráfica.

B).- Habilitar el puerto de comunicaciones con parámetros iniciales conocidos.

DESCRIPCION : Una vez que el puerto de comunicaciones es detectado y que la autoprueba resulte satisfactoria, éste deberá ser habilitado con los parámetros iniciales convenidos de antemano. Deberá notificarse al usuario el valor de dichos parámetros iniciales (Bauds, bits de dato, tipo de paridad y bits de paro) en una ventana gráfica permitiendo la modificación de cualquiera de ellos.

C).- Permitir la configuración de los parámetros del puerto de comunicaciones.

DESCRIPCION : Una vez que el puerto se encuentra en operación, existe la posibilidad de que algún parámetro del puerto requiera ser cambiado. Los parámetros configurables son:

- BAUDS.
- NUMERO DE BITS DE DATO.
- TIPO DE PARIDAD A MANEJAR.
- NUMERO DE BITS DE PARO.

Las opciones de dichos parámetros son mostrados en ventanas y menúes gráficos.

D).- Permitir la transmisión y recepción de información mediante peticiones.

DESCRIPCION : La transmisión y recepción de información mediante peticiones se hará cuando el usuario o un procedimiento de software requiera comunicarse con el dispositivo externo en un momento determinado.

E).- Permitir la recepción de información en cualquier momento sin que se haga una petición (recepción asíncrona).

DESCRIPCION : La recepción de información *asíncrona* se refiere a aquella información que llega al puerto serie mientras se está realizando un proceso distinto al de la captación de esta información. Esta información tiene que ser atendida sin discriminación ni pérdida por ninguna razón. Debido a esto, el módulo de comunicaciones deberá tener una rutina que le permita rescatar dicha información y almacenarla en memoria.

F).- Permitir verificar el estado del puerto en cualquier instante.

DESCRIPCION : La verificación del estado del puerto es realizada cuando se desee conocer las condiciones actuales del puerto y saber si existe alguna anomalía en el mismo, como por ejemplo la ausencia del conector. Los resultados de esta verificación son mostrados al usuario en una ventana gráfica.

G).- Permitir al usuario interactuar con el sistema mediante ventanas y menúes gráficos.

La implementación de este requerimiento se encuentra implícito en los puntos anteriores al hacer la especificación de ventanas y menúes gráficos presentados al usuario.

II.3.2 EL "COMO".

La segunda parte en el análisis de requerimientos consiste en determinar "cómo" se lograrán cumplir dichos requerimientos y con ello éstos son divididos en niveles de abstracción.

A).- Realizar una autoprueba al CI 8250 para verificar su operabilidad.

1er. NIVEL DE ABSTRACCION

MODULO DE AUTOPRUEBA: Este módulo deberá enviar cierta información al puerto para ser transmitida y recibida por él mismo. Una vez hecho esto, deberá interpretar la información y generar un reporte del resultado de esta operación indicando si el CI está operando correctamente o no. Como se puede observar resaltan tres funciones principales: Realizar la autoprueba, interpretar la información y generar reporte.

2° NIVEL DE ABSTRACCION:

MODULO REALIZA AUTOPRUEBA: Este módulo realizará la autoprueba sobre el CI 8250. La autoprueba consiste en configurar al 8250 en modo de loopback. Cuando un byte es puesto en el registro de transmisión de éste, es tomado por el transmisor y enviado al mismo tiempo a su registro de recepción.

MODULO REPORTE AUTOPRUEBA: Este módulo recibe la información generada por el módulo de autoprueba, interpretándola para determinar si el CI se encuentra en perfecto estado o si tiene alguna falla. El módulo no determina el tipo de falla que se encuentra en el CI, simplemente compara ambos datos (el enviado y el recibido) que deberán ser iguales para reportar el correcto funcionamiento del CI.

MODULO VENTANA AUTOPRUEBA: Este módulo recibe el reporte generado por el módulo anterior y genera una ventana con los resultados de la autoprueba, los parámetros con los que fue realizada, el dato transmitido y el dato recibido así como el resultado de la misma, indicando si el puerto se encuentra funcionando correctamente.

B).- Habilitar el puerto de comunicaciones con parámetros iniciales conocidos.

1er. NIVEL DE ABSTRACCION

MODULO PARAMETROS INICIALES: Cuando la autoprueba resulta satisfactoria, según un indicador del módulo anterior, el puerto de comunicaciones deberá ser configurado con ciertos parámetros iniciales. Estos parámetros son presentados al usuario en una ventana gráfica, y si desea cambiar algún(os) le serán presentadas las opciones de configuración del puerto en menúes gráficos. De ésto se desprenden dos funciones principales: La primera es presentar la configuración inicial del puerto de comunicaciones en una ventana y la segunda es permitir el cambio de alguna de las opciones de configuración del puerto.

2° NIVEL DE ABSTRACCION

MODULO VENTANA PARAMETROS INICIALES: Este módulo presenta los parámetros que serán puestos inicialmente al puerto de comunicaciones. El módulo cuenta con la opción para hacer un llamado al cambio de los parámetros. Si el usuario no desea cambiar algunos de los parámetros, este módulo debe habilitar la recepción asíncrona, en caso contrario el módulo siguiente de configuración se encargará de dicha habilitación.

MODULO CONFIGURA PUERTO: Este módulo es llamado cuando el usuario desea cambiar alguno de los parámetros iniciales del puerto. Este módulo es descrito en el siguiente inciso.

C).- Permitir la configuración de los parámetros del puerto de comunicaciones.

1er. NIVEL DE ABSTRACCION

MODULO CONFIGURA PUERTO: Este módulo permite modificar los parámetros

de operación del puerto de comunicaciones. Estos parámetros son: Bauds, bits de dato, paridad y bits de paro. Para ello, muestra al usuario menúes gráficos con las opciones para cada uno de los parámetros. Una vez que el usuario ha elegido los parámetros del puerto, estos le son presentados para que él corrobore dichos parámetros y sea configurado el puerto. De esto se desprenden las siguientes funciones: menúes para: bauds, dato, paridad, paro así como de una ventana con parámetros elegidos. Cuando se confirma la configuración del puerto, deberá inhibirse cualquier tipo de recepción (incluyendo la asíncrona) para evitar el recibir información durante un momento crítico de configuración del puerto.

2º NIVEL DE ABSTRACCION

Para cada uno de los módulos de este nivel, se genera un menú gráfico que muestra las opciones a elegir. Una vez que se tenga dicha elección, será puesta en una variable de configuración temporal.

MODULO DESHABILITA RECEPCION ASINCRONA: Como fue descrito en el inciso E del punto II.3.1, la información asíncrona llega en cualquier instante, mientras se realiza otro proceso. Para el caso de configuración, es necesario inhibir esta recepción debido a que se realizará un proceso de transición de parámetros; en tales condiciones, no se espera recibir información alguna.

MODULO MENU BAUDS: Este módulo permitirá elegir de entre las velocidades disponibles a la que operará el puerto: 110, 300, 600, 1200, 2400, 4800 ó 9600.

MODULO MENU DATO: Este módulo permite elegir el número de bits de dato que serán enviados por el canal de comunicaciones. Estos podrán ser 7 u 8 bits.

MODULO MENU PARIDAD: Este módulo permite elegir la paridad que será manejada durante la transmisión para propósitos de detección de errores. Las opciones son: PAR, IMPAR o NINGUNA.

MODULO MENU PARO: Este módulo permite elegir el número de bits de paro que serán empleados durante la comunicación para indicar el fin de la trama. Las opciones son 1 ó 2.

MODULO VENTANA PARAMETROS DE CONFIGURACION: Este módulo presenta, en una ventana gráfica, los parámetros con los que será configurado el puerto de comunicaciones. El módulo permite confirmar la configuración o bien regresar y corregir alguno de los parámetros. De ésto se desprenden dos funciones: poner los parámetros en el puerto y habilitar la recepción asíncrona.

3er. NIVEL DE ABSTRACCION

MODULO PONE BAUDS: Este módulo almacena el parámetro elegido del menú de bauds para la velocidad, en la configuración temporal del puerto. Para el caso de los bits de dato, paridad y bits de paro se realiza el mismo proceso y el nombre de los módulos son: MODULO PONE DATO, MODULO PONE PARIDAD y MODULO PONE PARO respectivamente.

MODULO PONE PARAMETROS: Una vez que el usuario ha confirmado los parámetros del puerto estos son obtenidos de la configuración temporal y colocados al CI 8250; habilitándose la recepción asíncrona.

MODULO HABILITA RECEPCION ASINCRONA: Una vez que el puerto es configurado, es necesario restablecer dicha recepción asíncrona para así continuar con el proceso normal del sistema.

D).- Permitir la transmisión y recepción de información mediante peticiones.

1°. NIVEL DE ABSTRACCION

MODULO TRANSMITE Y MODULO RECIBE: Estos módulos permiten la transmisión y recepción de cualquier tipo de información (caracter, cadena o archivo) cuando algún proceso de software lo requiera. Cualquiera de las peticiones, transmite o recibe, primeramente deberá inhibir la recepción asíncrona, posteriormente deberá verificar la existencia del conector para asegurar la transferencia de información. Una vez completada la transferencia, deberá habilitarse nuevamente la recepción asíncrona. De esto se desprenden tres funciones: deshabilitar la recepción asíncrona, verificar la existencia del conector y habilitar nuevamente la recepción asíncrona.

2° NIVEL DE ABSTRACCION

MODULO DESHABILITA RECEPCION ASINCRONA: Cuando se intenta realizar alguna operación con el puerto serie como lo son: configurar el puerto, recibir o enviar información tomando un control directo del puerto o bien verificar si existe conector, la función de recepción asíncrona interferirá en dichas operaciones, provocando en algunas ocasiones la perdida de control del puerto. por ello es necesaria su inhibición (de la cual se encarga este módulo) mientras se realiza la operación.

MODULO VERIFICA CONECTOR: Este módulo examina la existencia del conector para con ello asegurar en cierto grado la transferencia de información. Este módulo no verifica la existencia del dispositivo externo en el otro extremo del conector; presupone esta conexión y sólo requiere que el conector se encuentre presente en el puerto serie de la PC. Para llevar a cabo dicha verificación, se necesitan unir los pines de las señales DTR y DSR en el conector de la PC.

MODULO HABILITA RECEPCION ASINCRONA: Una vez que alguno de los procesos de recepción, transmisión, configuración o verificación de conector hayan culminado, se habilita nuevamente la recepción asíncrona para seguir recuperando la información que por el puerto llegue.

E).- Permitir la recepción de información en cualquier momento sin que se haga una petición.

1er. NIVEL DE ABSTRACCION

MODULO RECEPCION ASINCRONA: Este módulo, como se mencionó anteriormente, recibe toda aquella información que llega al puerto sin haber hecho una petición y la almacena en un espacio de memoria con la finalidad de no perderla y procesarla posteriormente.

F).- Permitir verificar el estado del puerto en cualquier instante.

1er. NIVEL DE ABSTRACCION

MODULO ESTADO PUERTO: Este módulo trabaja directamente con los registros del CI 8250 ya que lee los registros MODEM STATUS REGISTER y LINE STATUS REGISTER y con ello deducir la condición actual del puerto. La información que es mostrada al usuario es la siguiente:

- Si existe conector.
- Si existe la posibilidad de transmitir.
- Si existe dato en el puerto.
  - Si el dato fue sobrescrito.
  - Si el dato tiene un error de paridad.
  - Si el bit de paro fue erróneo.
  - Si se recibió una trama de ceros.

De esto se desprenden dos funciones: leer los registros del puerto para determinar el estado del puerto y generar una ventana gráfica mostrando dicho estado.

2° NIVEL DE ABSTRACCION

MODULO LEE REGISTROS: Este módulo verifica la existencia del conector y lee los registros del CI 8250 para con ello generar un reporte sobre el estado actual del puerto.

MODULO VENTANA ESTADO ACTUAL DEL PUERTO: Este módulo toma el reporte generado por el módulo anterior, crea una ventana gráfica mostrando dicho reporte al usuario.

Con ésto se concluye el análisis de los requerimientos. La siguiente etapa nos mostrará el diagrama final que tiene el módulo de comunicaciones; este diagrama depende de la metodología de diseño de software a usar.

II.4 TERCERA ETAPA: ELECCION DE LA METODOLOGIA DE DISEÑO.

Se propuso la metodología orientada a la estructura[h] de datos porque está enfocada hacia los sistemas modulares que tienen una gran flexibilidad para modificar o anexar funciones o procedimientos con un mínimo de cambios. Con ésta filosofía es fácil incorporar el módulo de comunicaciones a cualquier sistema y realizar cambios según nuevos requerimientos.

Esta metodología propone los siguientes pasos[i]:

1° Deberá realizarse un diagrama ENTRADAS - PROCESO - SALIDAS o IPO[j] (de sus siglas en inglés Input-Process-Output) con el fin de identificar cada una de estas partes y delimitar claramente las fronteras entre el proceso principal y el procesamiento de entradas y salidas (partes esenciales del diagrama IPO) que muchas veces no son fáciles de discernir.

2° Deberá realizarse un diagrama esquemático y JERARQUICO de los diferentes niveles que tendrá el software.

3° Anexo al esquema jerárquico se define una tabla de entradas y salidas que muestra los parámetros que fluyen a través del sistema.

4° Junto a la tabla de entradas y salidas deberá especificarse la función que realizará cada módulo, en lo que se denomina DICCIONARIO DE DATOS[b].

El diagrama de ENTRADAS - PROCESO - SALIDAS es el siguiente:

Figura II.1 Sistema IPO.

En la figura II.2 se muestra el esquema final del sistema o módulo de comunicaciones; además de la figura, se muestra a continuación el diccionario de datos y su tabla de entradas-salidas, como resultado de los pasos anteriores.

TABLA DE ENTRADAS Y SALIDAS.

NUMERO	ENTRADAS	SALIDAS
1	Identificador de recepción	bufer de recepción, error (por conector o por rebasar el máximo de caracteres)
2, 12, 13	ninguna	ninguna
3	tamaño, cadena	error (por conector, porque existe dato en el puerto, por error de paridad en éste último o bien por sobrescritura del mismo)
4, 9, 20	ninguna	deshabilitación correcta o incorrecta
5, 10	ninguna	ausencia o presencia de conector
6, 11, 31	ninguna	error al no poder realizar la habilitación
7	bandera de puerto disponible	ninguna
8	ninguna	bandera de puerto disponible
14	autoprueba	ninguna
15	resultado de autoprueba	ninguna
16	dato transmitido, dato recibido	resultado de autoprueba
17	dato transmitido	dato recibido
18	ninguna	reporte del estado

NUMERO	ENTRADAS	SALIDAS
19	reporte del estado	ninguna
21, 22, 23, 24	configuración	configuración
25	configuración	ninguna
26	identificador de bauds, configuración	configuración
27	identificador de datos, configuración	configuración
28	identificador de paridad, configuración	configuración
29	identificador de paro, configuración	configuración
30	configuración	ninguna

DICCIONARIO DE DATOS:

En el diccionario de datos se detalla cada uno de los módulos que se encuentran en el diagrama jerárquico. Aunque durante la descripción funcional de los módulos no importa el orden, se describirá cada módulo seguido de todas sus partes según el diagrama jerárquico.

RECEPCION ASINCRONA: Esta es una función que es independiente a cualquier otra, recibendo únicamente la información que llega al puerto de comunicaciones, cuando se esté realizando otro proceso diferente a éste, y colocando dicha

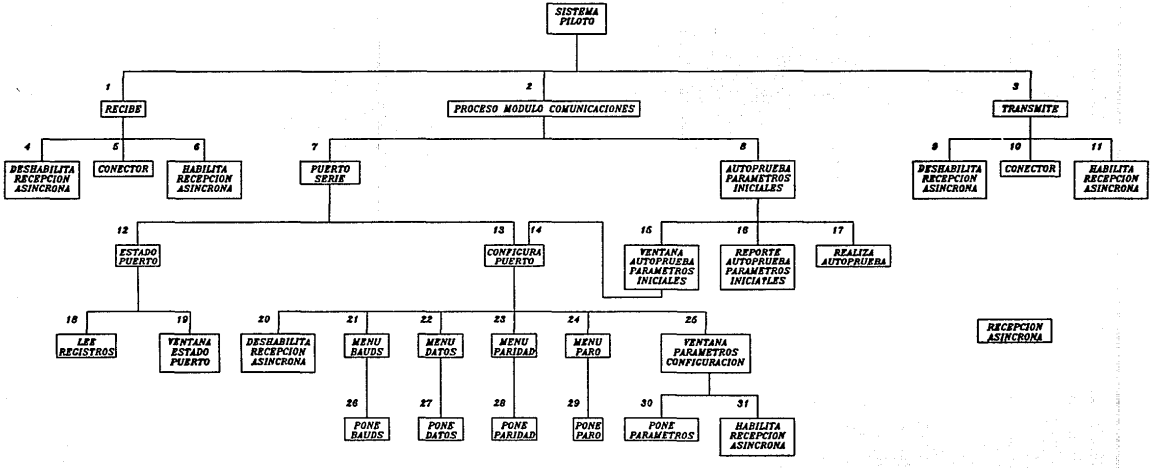

Figura II.2 Sistema terminal.

información en memoria. Esta información es referenciada bajo el nombre de BUFER DE RECEPCION ASINCRONA y recuperada como caracteres ASCII.

RECIBE: Esta función permite la recepción por petición, recibiendo la variable IDENTIFICADOR DE RECEPCION, que es una bandera para determinar el tipo de información que se recibirá y que puede ser:

- Byte
- Cadena
- Archivo

La recepción termina cuando llega al puerto un caracter que indica el fin de recepción ya sea de cadena o de archivo (para el caso de byte no es necesario dicho caracter de fin de recepción). Este caracter se determina en base al sistema operativo, por ejemplo para el sistema UNIX el fin de archivo está determinado por el caracter ASCII número 4, mientras que para el sistema DOS dicho caracter corresponde al número 26 en la tabla ASCII. Para el caso de recibir un caracter no se requiere indicar el fin debido a que se sabe la cantidad de información que se recibirá.

Si no llega el caracter de fin de recepción, para los casos cadena y archivo, la función permanecerá allí hasta recibirlo.

La información así recibida es identificada por una variable denominada BUFER DE RECEPCION, pudiendo ser recuperarada como caracteres en código ASCII.

Por otra parte, la función retorna un indicador de dos posibles ERRORES: cuando no existe conector o por exceso de caracteres recibidos. Para este último error se estableció un máximo de 256 caracteres.

Esta función hace un llamado a las siguientes tres funciones.

a).- DESHABILITA RECEPCION ASINCRONA: Esta función inhibe el proceso de la RECEPCION ASINCRONA con el fin de tomar un control total del puerto. Esta función regresa un código para indicar si la acción fue exitosa o no, en este último caso el código es un ERROR.

b).- CONECTOR: Esta función únicamente verifica la existencia de un conector en el puerto de comunicaciones. Para ello se activa la señal DTR recibiéndola por DSR. La función sólo retorna un valor indicando si existe o no dicho conector

c).- HABILITA RECEPCION ASINCRONA: Esta función reanuda la operación de RECEPCION ASINCRONA inhibida por la penúltima función (inciso a). Retorna un ERROR en caso de no poder realizar dicha operación.

TRANSMITE: Esta función permite enviar alguno de los siguientes tres tipos de información.

- Byte.
- Cadena.
- Archivo.

No es necesario especificarle el tipo de información que se requiere enviar ya que basta con los dos parámetros que le son pasados para que lleve a cabo dicho envío. Estos parámetros son:

° TAMAÑO
° CADENA

El parámetro TAMAÑO especifica, como su nombre lo indica, el tamaño en bytes de la información que se transmitirá. El segundo parámetro, CADENA, contiene la información que será enviada. Si el tamaño es mayor que la información contenida

en la cadena, se transmitira la información de CADENA primeramente, después se transmitira basura hasta llegar a los TAMAÑO bytes transmitidos. Puede no llevarse a cabo la transmisión si ocurre alguno de los siguientes errores que podrán ser reportados:

- ° Error al poner en modo de transmisión al puerto.
- ° Error al no disponer del conector.
- ° Error por tener un dato en el registro de recepción. Dentro de este error pueden suceder cualquiera de los siguientes condiciones que serán reportadas también como errores.
  - ° Dato sobrescrito.
  - ° Dato no válido por: Paridad erronea, recepción de ceros en toda la trama o bien por no recibir el bit de paro.

Esta función hace un llamado a las siguientes funciones que fueron descritas en la función RECIBE.

a).- DESHABILITA RECEPCION ASINCRONA.
b).- CONECTOR.
c).- HABILITA RECEPCION ASINCRONA

PUERTO SERIE: Esta función, en base al parámetro PUERTO DISPONIBLE, genera un botón gráfico que permitirá al usuario el acceso a las funciones de ESTADO PUERTO y CONFIGURA PUERTO. Este parámetro tiene un valor en base al resultado del proceso de autoprueba descrito más adelante.

Si el parámetro antes mencionado afirma que el puerto está presente y que se encuentra funcionando correctamente, entonces permite el acceso a un menú, con las opciones de ESTADO PUERTO y CONFIGURA PUERTO, cuando el usuario seleccione este botón gráfico.

ESTADO PUERTO: Esta función únicamente hace un llamado en secuencia a las siguientes dos funciones.

a).- LEE REGISTROS: Esta función primeramente verifica la existencia del conector, posteriormente hace una lectura a los registros LSR (Line Status Register) y MSR (Modem Status Register) del C.I 8250 y determina las condiciones actuales del puerto. Con esta información se genera un REPORTE DEL ESTADO que es retornado como parámetro.

b).- VENTANA ESTADO PUERTO: Esta función recibe el parámetro REPORTE DEL ESTADO y genera una ventana gráfica para mostrar al usuario el estado del puerto serie.

CONFIGURA PUERTO: Esta función primeramente lee los parámetros del puerto para almacenarlos en una variable de configuracion temporal, posteriormente genera un menú gráfico con las opciones de: BAUDS, DATO, PARIDAD, PARO y CONFIRMA. Cada una de estas opciones del menú hace un llamado a las siguientes funciones: MENU BAUDS, MENU DATO, MENU PARIDAD, MENU PARO y VENTANA PARAMETROS CONFIGURACION respectivamente.

Esta función únicamente recibe el parámetro AUTOPRUEBA, cuando es llamada por la función VENTANA AUTOPRUEBA PARAMETROS INICIALES (como se puede observar en el diagrama jerárquico). Con esta condición (parámetro AUTOPRUEBA), la función CONFIGURA PUERTO no hará un llamado a la función DESAHBILITA RECEPCION ASINCRONA, debido a que aún no ha sido activada dicha recepción asíncrona.

MENU [nombre]: Existen 4 funciones con el prefijo MENU seguido de uno de los siguietes nombre: BAUDS, DATO, PARIDAD o PARO. Estas 4 funciones realizan la misma operación. Dicha operación consiste en generar un menú gráfico con las opciones para cada nombre descrito; estas opciones son las siguientes:

NOMBRE	OPCIONES
BAUDS	150, 300, 600, 1200, 2400, 4800 y 9600
DATO	7 bits de dato 8 bits de dato
PARIDAD	PAR IMPAR NINGUNA
PARO	1 de paro 2 bits de paro

Estas función reciben la configuración actual del puerto en una variable denominada CONFIGURACION, determinan el valor de la opción seleccionada, hacen un llamado a la siguiente función descrita y regresan la variable CONFIGURACION modificada.

PONE [nombre]: Estas funciones son llamadas por su función correspondiente MENU [nombre]. Estas funciones tienen como prefijo PONE seguidas de uno de los nombres: BAUDS, DATO, PARIDAD y PARO. Estas funciones reciben como parámetros un IDENTIFICADOR : DE BAUDS, DE DATO, DE PARIDAD o DE PARO así como la CONFIGURACION actual del puerto con el fin de modificar esta última en base al primer parámetro. Las funciones regresan la nueva CONFIGURACION para el puerto de comunicaciones.

VENTANA PARAMETROS CONFIGURACION: Esta función genera una ventana gráfica mostrando los parámetros de operación del puerto serie. Estos parámetros son tomados de la CONFIGURACION, que es una entrada a la función. Esta función hace un llamado a dos funciones para completar la acción de configurar el puerto, estas funciones son: HABILITA RECEPCION ASINCRONA que ya fue descrita y PONE PARAMETROS que será descrita a continuación.

PONE PARAMETROS: Esta función recibe la variable CONFIGURACION como parámetro y con ello configurar el puerto de comunicaciones. La función no regresa ningún indicador sobre el proceso de la configuración.

AUTOPRUEBA PARAMETROS INICIALES: Esta función hace un llamado a tres funciones en la secuencia que a continuación se mostrará. La función regresa una bandera de PUERTO DISPONIBLE.

REALIZA AUTOPRUEBA: Esta función recibe como parámetros las variables DATO TRANSMITIDO y DATO RECIBIDO; pone en modo loopback (realimentación) al 8250 para realizar la autoprueba. Después de ésto, se envía la información que se encuentra en DATO TRANSMITIDO, se recibe dicha información para ser almacenada en DATO RECIBIDO. Cuando termina este proceso, el 8250 es regresado a operación normal.

REPORTE AUTOPRUEBA PARAMETROS INICIALES: Esta función recibe la información almacenada en las variables DATO TRANSMITIDO y DATO RECIBIDO para con ello generar un RESULTADO DE AUTOPRUEBA. A dicho resultado se le agrega, en caso de que se determine un correcto funcionamiento del puerto, los parámetros a los que inicialmente operará el puerto. En esta función tambien se determina el valor de la bandera PUERTO DISPONIBLE. El RESULTADO DE AUTOPRUEBA es regresado como parámetro.

VENTANA AUTOPRUEBA PARAMETROS INICIALES: Esta función toma el valor de RESULTADO DE AUTOPRUEBA como parámetro de entrada y genera una ventana gráfica para reportar el resultado de la autopreba. Si la bandera de PUERTO DISPONIBLE indica un correcto funcionamiento del puerto, esta función permite al usuario cambiar los parámetros iniciales mediante una llamada a la función CONFIGURA PUERTO. Cuando esta última es invocada, se le envía el parámetro AUTOPRUEBA para indicarle cual fue el motivo de la llamada.

II.5 CUARTA ETAPA: CODIFICACION.

El proceso de codificación consiste en generar un programa usando un lenguaje de programación, manteniendo un estilo de programación.

El lenguaje de programación para este caso en particular es lenguaje C y para generar las ventanas y menúes gráficos se tiene como herramienta el X-WINDOW SYSTEM.

El lenguaje C y X-WINDOW es debido a que existe un cierto porcentaje de código para el sistema piloto en éstos. Para facilitar la integración del módulo de comunicaciones a este código del sistema piloto, se emplea igualmente lenguaje C y X-WINDOW. Sin embargo, tanto C como X-WINDOW tienen razones aún más poderosas que son resaltadas cuando se discuta el ambiente de programación del punto II.5.1.

Otra razón no menos importante es lo referente al manejo del puerto serie. El lenguaje C cuenta con funciones estándar que realizan operaciones específicas sobre los dispositivos de entrada/salida como lo es el puerto serie. Esta funciones son: open, close, write y read. Dichas funciones podrán aparecer en otros lenguajes de alto nivel; pero, como se muestra en la sección "II.5.1 Ambiente de programación", tiene un aspecto más importante que el de formar parte del lenguaje: Son llamadas al sistema operativo.

Ahora bien, para cumplir con el punto de estilo de programación, se tomó el patrón de escribir los nombres de las funciones en español y los nombres de las variables y constantes en inglés.

Cabe aclarar que para este punto, no se presenta la codificación final, ésta es mostrada en el apéndice A; sin embargo, a continuación se presenta un pseudocódigo con propósitos de mostrar el procedimiento de cada función del código final.

CONSTANTES GLOBALES

```
COM2 = "/dev/ttyO1" /* ARCHIVO DESCRIPTOR PARA COM2 */
D_L_L_COM2 = 2F8h /* DIRECCION DE DIVISOR LATCH LEAST PARA COM2 */
D_L_M_COM2 = 2F9h /* DIRECCION DE DIVISOR LATCH MOST PARA COM2 */
RX_REG_COM2 = 2F8h /* DIRECCION DEL REGISTRO DE RECEPCION PARA COM2
*/
TX_REG_COM2 = 2F8h /* DIRECCION DEL REGISTRO DE TRANSMISION PARA
COM2 */
L_C_REG_COM2 = 2FBh /* DIRECCION DE LINE CONTROL PARA COM2 */
M_C_REG_COM2 = 2FCh /* DIRECCION DE MODEM CONTROL PARA COM2 */
L_S_REG_COM2 = 2FDh /* DIRECCION DE LINE STATUS PARA COM2 */
M_S_REG_COM2 = 2FEh /* DIRECCION DE MODEM STATUS PARA COM2 */
INT_COM2 = 3 /* INTERRUPCION DE HARDWARE PARA COM2 */
ON = 1
OFF = 0
RX_DATA_RDY = 1 /* VALOR QUE INDICA DATO EXISTENTE EN RX_REG_COM2
*/
DATA_ERR = 1 /* ERROR EN LA TRANSMISION AL EXISTIR DATO EN
RX_REG_COM2 */
OVER_ERR = 2 /* ERROR DE SOBRESCRITURA EN LA RECEPCION */
PAR_ERR = 4 /* ERROR DE PARIDAD EN LA RECEPCION */
```

```
FRAM_ERR = 8 /* ERROR POR BIT DE PARO INCORRECTO EN RECEPCION */
BREAK_ERR = 10h /* ERROR AL RECIBIR CEROS O CONEXION ROTA */
TX_RDY =- 40h /* POSIBILIDAD DE TRANSMITIR */
RX_LOOP_ERR = -2 /* ERROR DE RECEPCION DURANTE LA AUTOPRUEBA */
CONN_ERR = 3 /* ERROR POR NO EXISTIR CONECTOR */
RX_BYTE = 1 /* IDENTIFICADOR PARA RECIBIR UN BYTE */
RX_STRING = 2 /* IDENTIFICADOR PARA RECIBIR UNA CADENA */
RX_FILE = 3 /* IDENTIFICADOR PARA RECIBIR UN ARCHIVO */
EOS_RX = 0 /* CARACTER DE FIN DE CADENA RECIBIDA */
EOF_RX = 4 /* CARACTER DE FIN DE ARCHIVO EN UNIX */
CHAR_MAX_ERR = -1 /* ERROR POR LLEGAR AL MAXIMO DE CARACTERES
ESPECIFICADO EN                    CHAR_MAX */
CHAR_MAX = 256 /* VALOR DEFINIDO EN LIMITS.H */
ENBL_IO_ERR = -1 /* ERROR AL HABILITAR EL CANAL DE ENTRADA SALIDA */
SET_PRI_ERR = -2 /* ERROR AL PONER DENTRO DE UNIX LA PRIORIDAD DEL
PROGRAMA */
GET_INT_ERR = -3 /* ERROR AL OBTENER EL VECTOR DE INTERRUPCIONES PARA
                    HABILITAR LA INTERRUPCION DEL PUERTO SERIE */
REL_INT_ERR = -4 /* ERROR AL ATRAPAR LA INTERRUPCION PARA DESHABILITAR
LA                  RECEPCION ASINCRONA */
CONN_PRES = 2 /* INDICADOR DE CONECTOR PRESENTE */
CONN_ERR = 3 /* INDICADOR DE NO CONECTOR */
PAR_FILDES_ERR = -4 /* ERROR AL PONER PARAMETROS EN EL ARCHIVO
DESCRIPTOR */
OPEN_FILDES_ERR = -3 /* ERROR AL ABRIR ARCHIVO DESCRIPTOR */
PATH_RX_FILE = "\usr\rick\basuras\tx_file.dat" /* ARCHIVO DE RECEPCION */
DB_7 = CS7 /* BITS DE DATO IGUAL A 7 */
DB_8 = CS8 /* BITS DE DATO IGUAL A 8 */
PAR_EVEN = PARENBL /* PARIDAD PAR */
PAR_ODD = PARENBL OR PARODD /* PARIDAD IMPAR */
NO_PAR = 0 /* SIN PARIDAD */
STP_1 = 0 /* UN BIT DE PARO */
STP_2 = CSTOPB /* DOS BITS DE PARO */
```

NOTA : Los valores de B110 hasta B9600 asi como CS7, CS8, PARENBL, PARODD y CSTOPB son valores definidos en el archivo cabecera tèrmios.h

VARIABLES GLOBALES

```
ENBL_COM = ON
buff_rx /* BUFER EN RAM PARA LA RECEPCION */
buff_rx_int /* BUFER EN RAM PARA LA RECEPCION ASINCRONA */
loopback /* VARIABLE PARA EL ESTADO DEL LOOPBACK */
termios_p.c_cflag /* VARIABLE PARA ALMACENAR LA CONFIGURACION TEMPORAL
*/
str_bauds, str_data, str_parity, str_stop /* VARIABLES QUE CONTIENE LA
                                             ELECCION DE CONFIGURACION PARA
                                             SER MOSTRADA EN LA VENTANA DE
                                             CONFIGURACION */
```

PSEUDOCODIGO DE LOS MODULOS

```
NOMBRE DE LA FUNCION: recibe
PARAMETRO DE ENTRADA: ID_RX  /* identificador de recepción */
PARAMETROS DE SALIDA: buff_rx /* buffer de recepción */
                      error /* errores :
                               ° CONN_ERR cuando no existe conector
                               ° CHAR_MAX_ERR cuando se recibe mas de
                                        CHAR_MAX caracteres */
```

```
VARIABLES LOCALES: num_char /* número de caracteres recibidos */
                   tot_char /* total de caracteres recibidos */
                   conn /* recibe el valor de la función conector */

INICIA FUNCION recibe
  FUNCION deshabilita_recepcion_asincrona
  conn = FUNCION conector
  SI1 conn = CONN_ERR
      reporta error = CONN_ERR
      sale de la función recibe
  FIN SI1
  CICLO INFINITO1
      SI2 existen datos en el puerto
        lee datos del puerto
        num_char = número de datos leídos
        tot_char = num_char
        en CASO1 de que ID_RX = RX_BYTE
            copiar en buff_rx el primer byte recibido
            leer cualquier otro dato que esté en el puerto
            FUNCION habilitar_recepción_asíncrona
        FIN CASO1
        en CASO2 de que ID_RX = RX_STRING
            copiar en buff_rx los num_char bytes recibidos
            SI3 el último byte recibido es EOS_RX
              FUNCION habilita_recepción_asíncrona
              sale función recibe
            FIN SI3
            CICLO INFINITO2
              lee datos del puerto
              num_char = número de datos leídos
              tot_char = num_char + tot_char
              concatena en buff_rx los num_char bytes leídos
```

```
        SI4 el último byte recibido es EOS_RX
            FUNCION habilita_recepción_asíncrona
            sale de la función recibe
        FIN SI4
        SI5 tot_char es mayor o igual a CHAR_MAX
            FUNCION habilita_recepcion_asinrona
            reporta error = CHAR_MAX_ERR
            sale de la función recibe
        FIN SI5
    FIN CICLO INFINITO2
FIN CASO2
en CASO3 de que ID_RX = RX_FILE
    abre archivo para escritura de PATH_RX_FILE
    copia en buff_rx los num_char bytes recibidos
    SI6 el último byte recibido es EOF_RX
      escribe en el archivo los tot_char bytes contenidos en buff_rx
      cierra el archivo
      FUNCION habilita_recepción_asíncrona
      sale de la función recibe
    FIN SI6
    CICLO INFINITO3
      lee datos del puerto
      num_char = número de datos leídos
      tot_char = num_char + tot_char
      concatena en buff_rx los num_char bytes recibidos
      SI7 el último byte recibido es igual a EOF_RX
          escribe en el archivo los tot_char bytes de buff_rx
          cierra el archivo
          FUNCION habilita_recepción_asíncrona
          sale de la función recibe
      FIN SI7
      SI8 tot_char es mayor o igual a CHAR_MAX
          escribe en en archivo los tot_byte de buff_rx
```

```
                limpia buff_rx
                tot_char = 0
            FIN SI6
        FIN CICLO INFINITO3
    FIN CASO3
  FIN SI2
FIN CICLO INFINITO1
FIN FUNCION recibe
```

NOMBRE DE LA FUNCION: deshabilita_recepción_asíncrona
PARAMETROS DE SALIDA: error /* errores:
° REL_INT_ERR no fue posible deshabilitar la recepción asíncrona
° 0 deshabilitación correcta */

```
INICIA FUNCION deshabilita_recepción_asíncrona
  deshabilita la señal que activa a la función recepción_asíncrona
  remueve la interrupción del puerto serie INT_COM2
  SI1 no fue posible remover la interrupción
      reportar error = REL_INT_ERR
      sale de la función deshabilita_recepción_asíncrona
  FIN SI1
  reporta error = 0
FIN FUNCION deshabilita_recepción_asíncrona
```

NOMBRE DE LA FUNCION: conector
PARAMETRO DE SALIDA: error /* errores:
° CONN_ERR conector no presente
° CONN_PRES conector presente */

```
INICIA FUNCION conector

  pone la señal DTR del registro M_C_REG_COM2 en 0
  lee el contenido del regsitro M_S_REG_COM2
  pone la señal DTR del registro M_C_REG_COM2 en 1
  lee el contenido del regsitro M_S_REG_COM2
  SI1 el contenido del M_S_REG_COM2 es igual a cero
      reporta error = CONN_ERR
      sale de la función conector
  FIN SI1
 reporta error = CONN_PRES
FIN FUNCION conector
```

```
      NOMBRE DE LA FUNCION: habilita_recepción_asíncrona
      PARAMETRO DE SALIDA: error /* errores:
                                    ° ENBL_IO_ERR error al abrir el
                                        canal de entrada-salida
                                    ° SET_PRI_ERR error al poner
                                        prioridad al proceso
                                    ° GET_INT_ERR error al obtener el
                                        vector de interrupciones */

INICIA FUNCION habilita_recepción_asíncrona
  abre el canal de entrada-salida
  SI1 existe error al abrir el canal
      reporta error = ENBL_IO_ERR
      sale de la función habilita_recepcion_asincroa
  FIN SI1
  pone prioridad al proceso /* prioridad del programa */
  SI2 existe error al poner la prioridad
```

```
      reporta error = SET_PRI_ERR
      sale de la función habilita_recepción_asíncrona
   FIN SI2
   pone la señal que activará la función recepción_asíncrona
   pone INT_COM2 en el vector de interrupciones
   SI3 existe error al poner la interrupción en el vector
      reporta error = GET_INT_ERR
      sale de la función habilita_recepción_asíncrona
   FIN SI3
FIN FUNCION habilita_recepción_asíncrona



      NOMBRE DE LA FUNCION: transmite
      PARAMETROS DE ENTRADA: size /* tamaño en bytes del dato a transmitir */
                             str /* dato a transmitir */
      PARAMETRO DE SALIDA: error /* errores:
                                    ° PAR_FILDES_ERR error al poner
                                      parámetros al archivo
                                      descriptor
                                    ° CONN_ERR error en el conector
                                    ° DATA_ERR dato en RX_REG_COM2
                                      ° OVER_ERR dato reescrito
                                      ° PAR_ERR dato erróneo */

      VARIABLES LOCALES: conn /* recibe el valor de la función conector */
                         line_status /* contiene el valor de L_S_REG_COM2 */
                         bit_tx /* contiene el bit 7 de line_status */
                         error /* reporta el error antes de la transmisión */

INICIA FUNCION transmite
   FUNCION deshabilita_recepción_asíncrona
   obtiene los parámetros del puerto
```

```
  pone en modo de transmisión al puerto serie con los mismos parámetros
  SI1 no fue posible poner en transmisión al puerto
      FUNCION habilita_recepción_asíncrona
      reporta error = PAR_FILDES_ERR
      sale de la función transmite
  FIN SI1
  conn = FUNCION conector
  SI2 conn = CONN_ERR
      pone en recepción al puerto con los mismos parámetros
      FUNCION habilita_recepción_asíncrona
      reporta error = CONN_ERR
      sale de la función transmite
  FIN SI2
  line_status = contenido del registro L_S_REG_COM2
  SI3 el bit 1 de line_status = RX_DATA_RDY
      reporta error = DATA_ERR
      SI4 el bit 2 de line_status = OVER_ERR
        reporte error = OVER_ERR
        SI5 el bit 3 de line_status = PAR_ERR o el bit 4 de line_status =
FRAM_ERR o bit5 de line_status = BREAK_ERR
            reporta error = PAR_ERR
        FIN SI5
      FIN SI4
      pone en recepción al puerto con los mismos parámetros
      FUNCION habilita_recepción_asíncrona
      sale de la función transmite
  FIN SI3
  MIENTRAS1 size sea diferente de cero
      pone el siguiente byte de str en TX_REG_COM2
    bit_tx = 0
      MIENTRAS2 bit_tx = 0
        lee el contenido de L_S_REG_COM2
        bit_tx = al bit 7 de L_S_REG_COM2
```

```
      FIN MIENTRAS2
      se mueve en uno la posición de str para poder transmitir el siguiente byte
      decrementa en uno la variable size
   FIN MIENTRAS1
   pone en recepción al puerto con los mismos parámetros
   FUNCION habilita_recepción_asíncrona
FIN FUNCION transmite
```

```
      NOMBRE DE LA FUNCION: autoprueba_parámetros_iniciales
      PARAMETROS DE SALIDA: ENBL_COM

      VARIABLES LOCALES: tx /* arreglo de caracteres que contiene la
                            información a enviar para la autoprueba */
                         rx /* arreglo de caracteres donde se recibe
                            la información de la autoprueba */
                         result /* Un arreglo de caracteres donde se coloca el
                            resultado de la autoprueba y los parámetros
                            iniciales del puerto */
```

```
INICIA FUNCION autoprueba_parámetros_iniciales
   tx = "Test LoopBack"
   FUNCION realiza_autoprueba con los parámetros tx y rx
   FUNCION reporta_autoprueba_parámetros_iniciales con los parámetros tx, rx y
result
   FUNCION ventana_autoprueba_parámetros_iniciales con el parámetro result
   pone una alarma para que dicha ventana sea habilitada a los 7 segundos
FIN FUNCION autoprueba_parámetros_iniciales
```

```
NOMBRE DE LA FUNCION: realiza_autoprueba
PARAMETROS DE ENTRADA: tx /* caracter a transmitir */
                       rx /* caracter en blanco */
PARAMETROS DE SALIDA: rx /* caracter recibido. El mismo que el parámetro
                         de entrada sólo que con la información
                         recibida */

VARIABLES LOCALES: count1 /* contador del número bytes transmitidos */
                   count2 /* contador del número de intentos */
                   bit_rx /* indicador de dato recibido */
                   l_c_r /* valor de line control register */
                   m_c_r /* valor de modem control regsiter */
                   d_l_l /* valor de divisor latch least */
                   d_l_m /* valor de divisor latch most */

INICIA FUNCION realiza_autoprueba
  d_l_l = 0Ch /* Valor para 9600 Bauds en la parte baja del latch divisor */
  d_l_m = 00h /* Valor para 9600 Bauds en la parte alta del latch divisor */
  l_c_r = 80h /* Valor para accesar los registros del latch divisor */
  m_c_r = 1Fh /* Valor para loopback en el registro de control de modem y con
                 las señales de DTR y RTS activas */
  count1 = 0
  count2 = 0
  abre el canal de entrada-salida para el acceso a los registros del 8250
  SI, existe error al abrir el canal de entrada-salida
      reporta ENBL_COM = ENBL_IO_ERR
      sale de la función realiza_autoprueba
  FIN SI,
  pone en L_C_REG_COM2 el valor de l_c_r
  pone en D_L_L_COM2 el valor de d_l_l
  pone en D_L_M_COM2 el valor de d_l_m
  pone en M_C_REG_COM2 el valor de m_c_r
```

```
  l_c_r = 1Fh /* parámetros de 8 bits de dato, paridad par y un bit de paro
                 durante la autoprueba */
  MIENTRAS1 la longitud de tc sea diferente de cero
      pone el byte de la posición de tx en TX_REG_COM2
      HACER1
        leer el contenido de L_S_REG_COM2
        bit_rx = bit 1 del contenido de L_S_REG_COM2
        incrementa en 1 a count2
      FIN HACER1 MIENTRAS bit_rx = 0 o count2 sea menor o igual a 100,000
      count2 = 0
      el de byte de la posición rx+count1 = RX_REG_COM2
      incrementa la posición de tx
      incrementa count1
  FIN MIENTRAS1
  tx = tx - count1 /* regresa tx a su posicion inicial */
  m_c_r = 0Fh /* valor para deshabilitar el modo de loopback */
  pone en M_C_REG_COM2 el valor de m_c_r
FIN FUNCION realiza_autoprueba


      NOMBRE DE LA FUNCION: reporta_autoprueba_parámetros_iniciales
      PARAMETROS DE ENTRADA: tx, rx y result
      PARAMETROS DE SALIDA: result /* diagnóstico de la autoprueba */

INICIA FUNCION reporta_autoprueba_parámetros_iniciales
  copia en result los parámetros con que se realizó la autoprueba
  SI1 ENBL_COM = ON
      concatena en result el dato transmitido y el dato recibido
      SI2 contenido de tx es igual al de rx
        concatena en result "Estado del puerto: bueno"
      FIN SI2
      sino ENTONCES1
```

```
        concatena en result "Estado del puerto: malo"
        ENBL_COM = RX_LOOP_ERR
        sale de la función reporta_autoprueba_parámetros_iniciales
      FIN ENTONCES1
  FIN SI1
  sino ENTONCES2 si ENBL_COM = ENBL_IO_ERR
      cancatena en result "Error al abrir el canal de entrada-salida"
      sale de la función reporta_autoprueba_parámetros_iniciales
  FIN ENTONCES2
  concatena en result los parámetros iniciales del puerto
FIN FUNCION reporta_autoprueba_parámetros_iniciales



    NOMBRE DE LA FUNCION: ventana_autoprueba_parámetros_iniciales
    PARAMETROS DE ENTRADA: result

INCIA FUNCION reporta_autoprueba_parámetros_inciales
  genera ventana gráfica con el texto result
  SI1 ENBL_COM = ON
    abre archivo descriptor COM2
    SI2 si no es posible abrir archivo descriptos COM2
      ENBL_COM = OPEN_FILDES_ERR
    FIN SI2
    sino ENTONCES1
      pone los parámetros iniciales en el puerto
      SI3 no es posible poner los parámetros
        ENBL_COM = PAR_FILDES_ERR
      FIN SI3
    FIN ENTONCES1
    SI4 ENBL_COM = ON
      habilita FUNCION configura_puerto con párametro YES /* YES es recibido
                                                          en la variable
```

```
                                                        loopback */
      FIN SI4
   FIN SI1
FIN FUNCION ventana_autoprueba_parámetros_iniciales



      NOMBRE DE LA FUNCION: puerto_serie
      PARAMETROS DE ENTRADA: ENBL_COM

INICIA FUNCION puerto_serie
   SI1 ENBL_COM = ON
      genera un botón con la etiqueta "puerto" en el menú principal
      pone un llamado a la función estado_configura para este botón
      limpia buff_rx_int
   FIN SI1
FIN FUNCION puerto_serie



      NOMBRE DE LA FUNCION: estado_configura

INICIA FUNCION estado_configura
   genera un menú gráfico con las opciones "estado del puerto" y "configura"
   pone un llamado a una función para cada opción /* "estado del puerto" ->
                                                     estado_puerto
                                                     "configura"  ->
                                                     configura_puerto */
FIN FUNCION estado_configura
```

NOMBRE DE LA FUNCION: estado_puerto

VARIABLES LOCALES: report

```
INICIA FUNCION estado_puerto
   report = FUNCION lee_registro
   FUNCION ventana_estado_puerto con el parámetro report
FIN FUNCION estado_puerto
```

NOMBRE DE LA FUNCION: lee_registros
PARAMETROS DE SALIDA: report

VARIABLES LOCALES: report
line_status
conn

```
INICA FUNCION lee_registros
   FUNCION deshabilita_recepción_asíncrona
   line_status = cotenido del registro L_S_REG_COM2 .
   conn = FUNCION conector
   FUNCION habilita_recepción_asíncrona
   SI1 conn = CONN_ERR
      concatena en report "No existe conector"
   FIN SI1
   SI2 en line_status existe BREAK_ERR
      concatena en report "Error por recibir trama de ceros"
   FIN SI2
   SI3 en line_status existe FRAM_ERR
```

```
      concatena en report "Bit de paro erróneo"
   FIN SI3
   SI4 en line_status existe OVER_ERR
      concatena en report "Dato sobrescrito"
   FIN SI4
   SI5 en line_status existe PAR_ERR
      concatena en report "Error de paridad"
   FIN SI5
   SI6 en line_status existe RX_DATA_RDY
      concatena en report "Dato en bufer rx"
   FIN SI6
   SI7 en line_status existe TX_RDY
      concatena en report "Posibilidad de transmitir"
   FIN SI7
   regresa report
FIN FUNCION lee_registros
```

```
      NOMBRE DE LA FUNCION: ventana_estado_puerto
      PARAMETRO DE ENTRADA: report

INICIA FUNCION ventana_estado_puerto
   genera ventana gráfica
   muestra el contenido de report
FIN FUNCION ventana_estado_puerto
```

NOMBRE DE LA FUNCION: configura_puerto
PARAMETRO DE ENTRADA: loopback

```
INICIA FUNCION configura_puerto
   termios_p.c_cflag = configuración del puerto /* Variable de configuración
                                                   temporal */
   SI1 loopback = YES
      CONF_LOOPBACK = ON
   FIN SI1
   sino ENTONCES1
      FUNCION deshabilita_recepción_asíncrona
   FIN ENTONCES1
   genera menú gráfico con las opciones: "BAUDS", "DATO", "PARIDAD", "PARO"
     y "CONFIRMA"
   pone una llamada a las funciones menú_bauds, menú_dato, menú_paridad,
      menú_paro y ventana_parámetros_configuracion a cada opcion,
      respectivamente
FIN FUNCION configura_puerto
```

NOMBRE DE LA FUNCION: menú_bauds

```
INICIA FUNCION menú_bauds
   genera menú gráfico con las opciones: "110", "150", "300", "600", "1200",
      "2400", "4800" y "9600"
   pone una llamada a la función pone_bauds enviando un identificador para cada
      opción
FIN FUNCION menú_bauds
```

NOMBRE DE LA FUNCION: pone_bauds
PARAMETRO DE ENTRADA: ID_BAUDS

VARIABLE LOCAL: filter = FFF0h /* Valor para eliminar los bauds anteriores y poder colocar los nuevos en la variable de configuración temporal */

```
INICIA FUNCION pone_bauds
   termios_p.c_cflag = termios_p.c_cflag AND filter /* Elimina con una AND (a
                                                       nivel de bits) los bauds
                                                       anteriores de la variable
                                                    de configuración temporal
                                                       */
   en CASO1 de que ID_BAUDS = B110
      termios_p.c_cflag = termios_p.c_cflag OR B110 /* Pone los nuevos Bauds con
                                                       una OR a nivel de bits en
                                                       la variable de
                                                       configuración temporal */
      str_bauds = "110"
   FIN CASO1
   en CASO2 de que ID_BAUDS = B150
      termios_p.c_cflag = termios_p.c_cflag OR B150
      str_bauds = "150"
   FIN CASO2
   en CASO3 de que ID_BAUDS = B300
      termios_p.c_cflag = termios_p.c_cflag OR B300
      str_bauds = "300"
   FIN CASO3
   en CASO4 de que ID_BAUDS = B600
      termios_p.c_cflag = termios_p.c_cflag OR B600
      str_bauds = "600"
   FIN CASO4
```

```
    en CASO5 de que ID_BAUDS = B1200
        termios_p.c_cflag = termios_p.c_cflag OR B1200
        str_bauds = "1200"
    FIN CASO5
    en CASO6 de que ID_BAUDS = B2400
        termios_p.c_cflag = termios_p.c_cflag OR B2400
        str_bauds = "2400"
    FIN CASO6
    en CASO7 de que ID_BAUDS = B4800
        termios_p.c_cflag = termios_p.c_cflag OR B4800
        str_bauds = "4800"
    FIN CASO7
    en CASO8 de que ID_BAUDS = B9600
        termios_p.c_cflag = termios_p.c_cflag OR B9600
        str_bauds = "9600"
    FIN CASO8
FIN FUNCION pone_bauds


        NOMBRE DE LA FUNCION: menú_datos

INICIA FUNCION menú_datos
    genera menú gráfico con las opciones: "7 bits" y "8 bits"
    pone una llamada a la función pone_datos enviando un identificador para cada
        opción
FIN FUNCION menú_datos
```

```
    NOMBRE DE LA FUNCION: pone_datos
    PARAMETRO DE ENTRADA: ID_DATA

    VARIABLE LOCAL: filter = FFCFh /* Valor para eliminar los datos anteriores
                              y poder colocar los nuevos en la variable
                                  de configuración temporal */

INICIA FUNCION pone_datos
  termios_p.c_cflag = termios_p.c_cflag AND filter /* Elimina con una AND (a
                                                   nivel de bits) los bauds
                                                   anteriores de la variable
                                                   de configuración temporal
                                                    */
  en CASO1 de que ID_DATA = DB_7
    termios_p.c_cflag = termios_p.c_cflag OR DB_7 /* Pone los nuevos datos con
                                                  una OR a nivel de bits en
                                                  la variable de
                                                  configuración temporal */
    str_data = "7"
  FIN CASO1
  en CASO2 de que ID_DATA = DB_8
    termios_p.c_cflag = termios_p.c_cflag OR DB_8
    str_bauds = "8"
  FIN CASO2
FIN FUNCION pone_datos



    NOMBRE DE LA FUNCION: menú_paridad

INICIA FUNCION menú_paridad
  genera menú gráfico con las opciones: "PAR" "IMPAR" o "NINGUNA"
```

pone una llamada a la función pone_paridad enviando un identificador para cada opción
FIN FUNCION menú_paridad

NOMBRE DE LA FUNCION: pone_paridad
PARAMETRO DE ENTRADA: ID_PARITY

VARIABLE LOCAL: filter = FCBFh

INICIA FUNCION pone_paridad
termios_p.c_cflag = termios_p.c_cflag AND filter
en $CASO_1$ de que ID_PARITY = PAR_EVEN
termios_p.c_cflag = termios_p.c_cflag OR PAR_EVEN
str_parity = "PAR"
FIN $CASO_1$
en $CASO_2$ de que ID_PARITY = PAR_ODD
termios_p.c_cflag = termios_p.c_cflag OR PAR_ODD
str_parity = "IMPAR"
FIN $CASO_2$
en $CASO_3$ de que ID_PARITY = NO_PAR
str_parity = "NINGUNA
FIN $CASO_3$
FIN FUNCION pone_paridad

NOMBRE DE LA FUNCION: menú_paro

INICIA FUNCION menú_paro
genera menú gráfico con las opciones: "1" o "2"
pone una llamada a la función pone_paro enviando un identificador para cada

opción
FIN FUNCION menú_paro

NOMBRE DE LA FUNCION: pone_paro
PARAMETRO DE ENTRADA: ID_STOP

VARIABLE LOCAL: filter = FFBFh

INICIA FUNCION pone_paro
termios_p.c_cflag = termios_p.c_cflag AND filter
en $CASO_1$ de que ID_STOP = STP_1
str_stop = "1"
FIN $CASO_1$
en $CASO_2$ de que ID_STOP = STP_2
termios_p.c_cflag = termios_p.c_cflag OR STP_2
str_stop = "2"
FIN $CASO_2$
FIN FUNCION pone_paro

NOMBRE DE LA FUNCION: ventana_parámetros_configuracion

VARIABLE LOCAL: str_param

INCIA FUNCION ventana_parámetros_iniciales
concatena en str_param los valores de str_bauds, str_data, str_parity y str_stop para presentar los valores de la configuración
crea una ventana gráfica con el texto de str_param
pone un botón "SI" que acepta los parámetros
pone una llamada a la función pone_parámetros para este botón
FIN FUNCION ventana_parámetros_iniciales

NOMBRE DE LA FUNCION: pone_parámetros

INICIA FUNCION pone_parámetros
pone los parámetros de la variable temporal termios_p.c_cflag en el archivo descriptor COM2
FIN FUNCION pone_parámetros

II.5.1 AMBIENTE DE PROGRAMACION.

El ambiente bajo el que se trabajó el módulo de comunicaciones fue el siguiente:

HARDWARE:

1.- Computadora personal 386.
2.- Monitor Super VGA a color.
3.- Mouse.
4.- Disco duro de 540 MB.
5.- 8 MB de memoria RAM.

SOFTWARE:

1.- Sistema Operativo VENIX V R3.2 (Versión de UNIX de Ventur Co.).
2.- X-WINDOW SYSTEM como manejador de ventanas y menúes gráficos.
3.- Compilador para lenguaje C.

Para este caso en particular, las especificaciones de hardware son básicamente impuestas por el sistema operativo a utilizar, debido a que la version de VENIX® que se adquirió fue para PC 386. La capacidad de disco requerida para el VENIX es de 120 MB mínima. La cantidad de memoria RAM es necesaria para el manejo de los menúes

y gráficas de alta resolución requeridos por la herramienta X-WINDOW SYSTEM, así como un mouse y un monitor Super VGA para explotar al máximo el manejo de las ventanas y la calidad de presentación que proporciona este sistema X-WINDOW.

El sistema VENIX se encuentra como plataforma de software propuesto del sistema piloto. El Venix proporciona ciertas características y ventajas por ser multiusuario y multitarea, otra ventaja adicional son las facilidades que tiene para crear un ambiente de red, lo que es muy útil para implementar un sistema distribuido basado en SCADA.

El lenguaje C, como se mencionó en la sección anterior, fue debido a que se tiene un cierto porcentaje del código del sistema piloto en este lenguaje. Sin embargo, existen razones aún más fuertes por las que el lenguaje C fue seleccionado:

Por naturaleza, C es el lenguaje de UNIX, por ello presenta un gran número de ventajas que no se encuentran disponibles para otros lenguajes. Algunas de estas ventajas son:

- Llamadas al sistema operativo con funciones implícitas dentro de C que facilitan este proceso (open, read, write e ioctl).

- Acceso a las señales que maneja el sistema operativo internamente.

- Funciones propias del lenguaje que permiten el mismo proceso de manejo que UNIX realiza sobre los dispositivos físicos.

Además de estas ventajas, el lenguaje C permite la modularidad (mediante funciones) y la fácil incorporación de rutinas dentro de cualquier otro código (mediante la directiva #include).

Los sistemas operativos UNIX maneja todos los recursos del sistema (monitor, impresora, teclado, puerto serie etc) como archivos y no como unidades físicas. Estos archivos son denominados **ARCHIVOS DESCRIPTORES**[1]. Para hacer un acceso a un

dispositivo periféricos es necesario accesar el archivo descriptor que corresponde a este dispositivo.

Para el caso del puerto serie, generalmente existen los archivos descriptores ttyO0 para COM1, tty01 para COM2 y así sucesivamente.

Existen tres formas para accesar el puerto serie de comunicaciones: mediante el shell del sistema operativo, programando streams[2] en lenguaje C o bien, programando llamadas al sistema desde lenguaje C.

Debido a que el módulo de comunicaciones es un programa en lenguaje C, la primera forma de accesar al puerto serie queda descartada. La segunda forma, programando streams, tiene ciertas limitantes en cuanto a que las peticiones de entrada/salida pasan primeramente por un bufer intermedio que hace más lento el acceso al driver del dispositivo.

La tercera forma, llamadas al sistema, son accesos directos al driver del dispositivo sin necesidad de un bufer intermedio. Estos drivers son trabajados directamente por el sistema operativo para el control de entrada/salida a los dispositivos físicos.

Estas llamadas al sistema son básicamente: *open*, *read*, *write* e *ioctrl*. La primera llamada, *open*, nos permite abrir el archivo descriptor que corresponde al puerto serie (o dispositivo que se desea accesar); sin esta apertura del archivo descriptor no es posible llevar a cabo las operaciones de entrada/salida sobre el puerto serie. La segunda y tercer llamada, *read* y *write*, permiten leer y escribir información al puerto serie, en otras palabras permiten la recepción y transmisión de información respectivamente. La última opción, *ioctrl*, es para leer y cambiar los parámetros del puerto serie.

---

[2] *Los STREAMS (o corrientes como algunos traductores lo manejan) son un conjunto de librerías de entrada/salida codificadas en lenguaje C para facilitar la transferencia de información entre el usuario y un dispositivo.*

Estas llamadas al sistema permiten un control parcial sobre el puerto serie debido a que sólo es posible recibir, transmitir y cambiar los parámetros del puerto serie, sin posibilidades de realizar la autoprueba del CI 8250 o leer algún registro de este último.

Para que la función de recepción asíncrona pudiera realizar su tarea, se tenían las siguientes opciones:

- Alarmas.
- Señales.
- Polling.

Cada una de ellas presenta ventajas y desventajas, mismas que se describe a detalle en el siguiente capítulo basándose en los resultados de las pruebas realizadas.

Todo esto provocó el tomar en cuenta un recurso que viene incluído en la versión VENIX V R3.2 : Rutinas en tiempo real.

Estas rutinas en tiempo real permiten abrir el canal de entrada-salida y accesar el dispositivo directamente sin necesidad del archivo descriptor, así como trabajar directamente con los registros del CI 8250 sin ninguna restricción.

Cuando ya se tenía el camino a seguir para controlar totalmente el puerto serie, sólo restaba la programación de las ventanas y menúes gráficos.

Esta última parte se logró con la herramienta X-WINDOWS, ya mencionada, que permite crear una INTERFAZ GRAFICA de USUARIO (Graphical User Interface o GUI para abreviar). Esta GUI permite la interacción entre el usuario y el módulo de comunicaciones en un ambiente completamente gráfico.

Este sistema X-WINDOWS™ se encuentra disponible en algunas versiones de UNIX y tiene como características principales:

a).- Aplicación sobre sistemas a nivel industrial: Este sistema es uno de los más empleados para la creación de GUI's en plantas industriales.

b).- Permite menjo de multitareas: Este sistema permite el trabajar con varios procesos al mismo tiempo.

c).- Implicitamente permite el manejo de red: Debido a que UNIX es plataforma de X-WINDOW, éste también permite la operabilidad para trabajar con terminales.

d).- Explotar al máximo las facilidades de la red debido a que X-WINDOW soporta el modelo CLIENTE-SERVIDOR. Este modelo, en X-WINDOW, permite ejecutar el CLIENTE o el SERVIDOR en cualquier nodo de la red.

Mientras que algunas ventajas aplicables al módulo de comunicaciones son:

e).- Codificación en C. Las rutinas y funciones se encuentran en lenguaje C lo que permite su directa utilización en dicho módulo.

f).- X-WINDOW, al igual que UNIX, puede trabajar bajo cualquier plataforma de hardware, lo cual permite su portabilidad a cualquier sistema.

g).- Tiene una estructura orientada a objetos, lo que facilita el empleo de las rutinas y reduce grandemente el código fuente.

X-WINDOW genera ventanas con los mismos atributos que el sistema de ventanas WINDOWS para DOS: permite maximizar y minimizar la ventana, mover la ventana así como tener una barra vertical y horizontal para realizar el scroll sobre la

ventana. Todo ésto permite que el usuario que está relacionado con el manejo de ventanas en WINDOWS de DOS le sea familiar el manejo de ventanas en UNIX.

La unión de las funciones de X-WINDOW, llamadas al sistema y las de tiempo real permiten la operabilidad del módulo de comunicaciones. Con estas unión de funciones finalmente se encuentra codificado dicho módulo.

CAPITULO III

PRUEBAS

INTRODUCCION

Una vez que se ha generado el código, comienzan las pruebas del programa. Estas se hacen con el fin de comprobar el correcto comportamiento del software desarrollado y el cumplimiento total de los requerimientos planteados.

Dichas pruebas no solo pretenden comprobar que el código generado está cumpliendo con la función definida para el mismo, sino que permite la depuración de éste buscando y validando algunas otras opciones más viables. Con ello se puede observar que el proceso de desarrollo de software es un método iterativo y dinámico para la busqueda de la mejor solución y el cumplimiento de los requerimientos.

Por otra parte, para realizar las pruebas era necesario tener un dispositivo externo con el cual se pudiera llevar a cabo la comunicación y poder validar el módulo de comunicaciones. Para resolver este problema, se usó como dispositivo externo otra PC trabajando en sistema operativo DOS. Por ello, el capítulo se encuentra dividido en:

° Pruebas entre dos PC's con sistema operativo DOS en ambas computadoras.

° Pruebas entre dos PC's, una con sistemas operativos UNIX (módulo de comunicaciones) y la otra con sistema operativo DOS (dispositivo externo).

° Y por último, las pruebas entre una PC con sistema operativo UNIX (módulo de comunicaciones) y un dispositivo externo (no PC).

III.1 PRUEBAS ENTRE 2 PC's CON SISTEMA OPERATIVO DOS.

Las primeras pruebas que se realizaron de una comunicación serie asíncrona empleando RS-232 fueron entre dos PC's con sistema operativo DOS.

Dichas pruebas se hicieron necesarias debido a que, con ello se tendría el sistema que validara las pruebas que se harían al módulo de comunicaciones. También era necesario tener algún sistema que llevara a cabo esta comunicación con la característica de monitorear el proceso. Por razones de sencillez, fue más fácil implementar y validar, primeramente, un sistema en PC bajo DOS (llamémosle en adelante PC-DOS) y con ello poder validar posteriormente el módulo de comunicaciones en la PC bajo UNIX (llamémosle PC-UNIX).

La estrutura general del software para las pruebas entre dos PC-DOS es la mostrada en la figura III.1

Como se puede observar en dicha figura, se utilizaron pantallas (en modo texto) para el mononitoreo del proceso de comunicaciones (transmisión, recepción, estado y configuración). Este sistema en PC-DOS permite las siguientes funciones:

- Configurar el puerto.
- Realizar una autoprueba del puerto.
- Enviar.
- Recibir.
- Monitorear el estado del puerto.

Dicho sistema se implementó en lenguaje C, mientras que las rutinas para el control del CI 8250 se implementaron con lenguaje ensamblador del 80x86. El lenguaje C se empleó para las rutinas de ventanas y menúes que permiten al usuario interactuar con el sistema. El lenguaje ensamblador fue utilizado para las rutinas que trabajan directamente con el CI 8250". Estas últimas fueron:

- num_com_asm.
- configura_com_asm.
- loopback_com_asm.
- tx_com_asm.
- rx_com_asm.
- estado_com_asm.

A continuación se da una descripción funcional de estas rutinas.

- num_com_asm: En las PC existe un registro de 16 bits en el cual cada bit indica qué dispositivos se encuentran instalados en el sistema; siendo los bits 9, 10 y 11 destinados para especificar el número de puertos serie disponibles en el mismo. Para leer dicho registro, basta con invocar la interrupción 11h del BIOS. La función num_com_asm hace un llamado a esta interrupción y retorna así los dispositivos instalados en el sistema, como un entero, para después extraer el número de puertos serie instalados.

- configura_com_asm: Esta función permite configurar el puerto de comunicaciones seleccionado, con los parámetros elegidos por el usuario. Esta función recibe el número de puerto seleccionado (COM1 ó COM2) y la configuración que tendrá éste. Estos parámetros son variables de tipo entero. Dicha función no retorna ningún valor.

- loopback_com_asm: Esta función permite realizar la autoprueba del CI 8250 del puerto especificado.

- tx_com_asm: Esta función permite transmitir cierta cantidad de bytes especificados, de una localidad de memoria, por el puerto seleccionado.

- rx_com_asm: Esta función permite recibir por el puerto seleccionado, cierta cantidad de bytes y colocarlos en una localidad de memoria específica.

- estado_com_asm: Esta función permite leer los registros de estado del 8250 para así conocer el estado del puerto.

Miestras que la descripción funcional del programa es:

1° Determina el número de puertos serie disponibles. Para ello hace un llamado a la función num_com_asm.

2° Muestra un menú que permite seleccionar el puerto serie con el que se trabajará durante la comunicación.

3° Una vez elegido el puerto, se presenta una ventana con los parámetros iniciales que éste tendrá. Los parámetros son puestos mediante la función configura_com_asm. En caso de alguna modificación, se hará mediante la opción de configuración del menú principal.

4° Se genera una pantalla con un menú principal en la parte superior con las siguientes opciones:

CONFIGURA: Esta opción permite configurar los parámetros del puerto serie (Bauds, Bits de dato, Paridad, Bits de paro) a través de la función configura_com_asm así como poder realizar la autoprueba del CI 8250 mediante la función loopback_com_asm.

ENVIA: Esta opción presenta una ventana que permite definir la información a enviar (Caracter, Cadena o Archivo) y posteriormente introducir dicha información[1]. La información es enviada con la función tx_com_asm y en caso de ocurrir algún error éste será mostrado en una ventana.

RECIBE: Esta opción primeramente muestra una ventana que permite capturar el número de bytes de información a recibir como mínimo o bien cuando por el puerto arribe el caracter NULO[2]. Para llevar a cabo la recepción, esta opción hace un llamado a la función rx_com_asm.

MONITOREA: Esta opción hace un llamado a la función estado_com_asm para recibir de ésta el valor de los registros de estado del modem y de la línea (MODEM STATUS REGISTER y LINE STATUS REGISTER[4]) del CI 8250 y con ello presentar un reporte al usuario sobre el estado actual del puerto de comunicaciones

SALIR: Esta opción permite abandonar el sistema y regresar el control al DOS.

[1] Para el caso de Archivo, se introduce la ruta y el nombre del mismo

[2] Este caracter NULO corresponde en la tabla ASCII al 0.

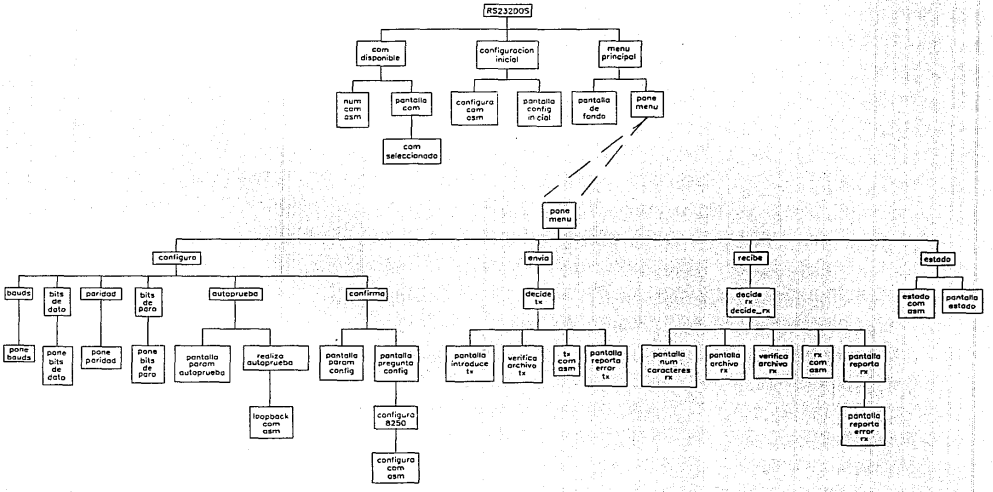

Figura III.1 Sistema de prueba entre PC's.

5° A partir de este momento se puede acceder a cualquier opción de este menú principal para realizar la operación deseada. Es posible abandonar alguna opciones mediante la tecla de Esc, antes de ésta realice su operación.

Con este módulo de comunicaciones y con un cableado entre dos PC-DOS que contempla un conector DB-9 para un puerto de una PC-DOS y un DB-25 para el otro (como se muestra a en la figura III.2) se realizaron las pruebas entre dos PC-DOS. La validación de las pruebas consistió en:

1° Configurar las PC con los mismos parámetros.

2° Verificar que el puerto se encuentre en condiciones de operación (monitoreando el estado del mismo).

3° Enviar y recibir información de manera HALFDUPLEX.

4° Eliminar el conector en alguno de los extremos y verificar que se reporta dicho error en este extremo cuando se intente realizar alguna de las operaciones de enviar o recibir.

5° Colocar nuevamente el conector y reintentar el 3er punto verificando que ya no es reportado el error.

6° Cambiar alguno de los parámetros en cualquiera de las PC y verificar que la información no es recibida correctamente.

7° Configurar nuevamente los parámetros de las PC y repetir desde el 3er punto.

Estos puntos fueron aplicados a las PC-DOS usadas para la validación de este software. Se realizaron varias pruebas al mismo puesto que fue necesaria una depuración del código y cambios en la presentación.

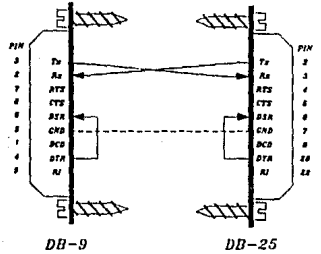

Figura III.2 Conexión DB-9 y DB-25.

Los obstáculos encontrados al realizar las pruebas de validación fueron:

1.- Pérdida del control del sistema cuando no llegaba el caracter NULO al final de la recepción.

Este problema parece limitar al sistema a sólo recibir información que contega dicho término NULO, sin embargo, el problema se resuelve al permitirle al usuario epecificar el número de caracteres a recibir para con ello asegurar que el caracter NULO pueda o no llegar.

2.- Pérdida de control del sistema o bien recepción de información que provocaba un diagnóstico erróneo sobre las condiciones del CI 8250 al momento de realizar el proceso de autuprueba. Durante las pruebas se observó que el comportamiento del CI al tenerlo en un estado de loopback era un poco diferente a lo que se tenía en las hojas de especificación del mismo. Este extraño comportamiento se encontraba en la siguiente secuencia de eventos:

° Se transmite un byte de información.

° Se verifica el bit del registro LINE STATUS REGISTER (LSR) que nos indica cuando un dato se encuentra disponible en el registro de recepción.

° Si este bit se encuentra en 1 lógico, entonces existe un dato en el registro de recepción.

° Inmediatamente se lee el byte del registro de recepción. Este byte debe corresponder al dato transmitido en caso de que no exista ninguna anomalía en el CI.

Sin embargo, todo resulta bien hasta el tercer punto ya que en el cuarto, la información recibida era errónea. Las pruebas realizadas sobre diferentes PC hicieron sospechar que existía una anomalía en el programa. Sin embrago, la lógica parecía correcta.

Las pruebas al enviar y recibir una cadena de bytes demostraron que la información se recibía truncada como si se estuviera transfiriendo información entre dos máquinas con algún parámetro diferente (Bauds, Bits de Dato, Tipo de Paridad o Bits de Paro), ya que en estos casos si se envía la palabra "HOLA" se puede recibir como "HL", perdiéndose información en la transferencia. Pero en el caso del modo loopback, sólo existe una configuración para el transmisor y el receptor.

Este hecho lleva a suponer que como el tiempo de acceso al canal de entrada/salida de la PC, por donde fluye la información de ésta hacia el CI 8250, depende básicamente del reloj que usa el microprocesador, así como una transmisión o recepción del 8250 depende de su propio reloj, un proceso de lectura o escritura de la PC (a través del canal de entrada/salida) es mucho más rápido que una transmisión o recepción del 8250.

Este problema se resolvió implementando en la rutina loopback_com_asm un contador para imponer un tiempo máximo en que debe arribar dicho dato. Si el dato leído no corresponde al dato transmitido al término de este contador, entonces el CI 8250 tiene un problema y es necesaria su revisión.

III.2 PRUEBAS ENTRE PC-UNIX Y PC-DOS.

Una vez que se tiene el sistema sobre el cual se validarán las pruebas del módulo de comunicaciones, el siguiente punto es probar dicho módulo de comunicaciones.

Para lograr cumplir el objetivo del capítulo fue necesario agregar al módulo de comunicaciones, algunas funciones que permitieran ver el desarrollo del proceso durante las pruebas. Por éste motivo, al módulo de comunicaciones se le incorporaron funciones que no estaban consideradas en el capítulo anterior pero que sin embargo se implementaron durante las pruebas. Este sistema se muestra en la figura III.3.

Su descripción es la siguiente:

modulo_de_comunicaciones: Esta función se encarga de generar un menú gráfico con las opciones de Recibir, Puerto Serie y Transmitir con el patrón siguiente de llamadas a funciones:

BOTON	FUNCION
Recibir	rx_CB
Puerto Serie	serial_port_CB
Transmitir	tx_CB

rx_CB: Esta función genera un menú gráfico con las opciones para recibir Caracter, Cadena o Archivo.

receive: Esta función tiene como parámetro de entrada al identificador ID_RX el cual le especifica el tipo de información a recibir

ID_RX = 1 (Byte o RX_BYTE)
ID_RX = 2 (Cadena o RX_STRING)
ID_RX = 3 (Archivo o RX_FILE)

Una vez identificada la información que será recibida, es llamada la función recibe para llevar a cabo el proceso de recepción.

win_results_Rx_CB: Esta función es la encargada de mostrar la información recibida en una ventana gráfica y los errores que durante ella ocurrieran.

serial_port_CB: Esta función se encarga de generar un menú gráfico con las opciones de Configurar y Monitorear el puerto serie. Las funciones que son llamadas al momento de activar alguno de estos botones son configura_puerto y estado_puerto respectivamente.

tx_CB: Esta función genera un menú gráfico con las opciones para transmitir Caracter, Cadena o Archivo.

win_tx_CB: Esta función toma como parámetro de entrada un identificador (ID_TX) para determinar el tipo de información que se va a transmitir.

ID_TX = 1 (Byte o TX_BYTE)
ID_TX = 2 (Cadena o TX_STRING)
ID_TX = 3 (Archivo o TX_FILE)

En base a este identificador, se genera una ventana para poder introducir la información a enviar de caracter o cadena o bien presenta una ventana con los archivos disponibles para su transmisión.

Tx_CB: Esta función recibe el mismo identificador de la función win_tx_CB para que en base a ésta se haga un llamado a la función transmite. Una vez ejecutada esta última, si existió algún error, será reportado en una ventana gráfica generada por ésta función.

Una vez implementadas estas funciones, ya es posible llevar a cabo las pruebas al módulo. Dichas pruebas consisten en los mismos puntos que fueron aplicados para las pruebas entre dos PC-DOS y que son:

1º Configurar PC-UNIX con los mismos parámetros de PC-DOS.

2º Verificar que el puerto de PC-UNIX se encuentre en perfectas condiciones (monitoreando el estado del mismo).

3º Enviar y recibir información de manera HALFDUPLEX.

4º Eliminar el conector en la PC-UNIX y verificar que se reporta dicho error cuando se intente realizar alguna de las opciones del 3er punto.

5º Colocar nuevamente el conector y reintentar el 3er punto verificando que ya no es reportado el error.

6º Cambiar alguno de los parámetros en cualquiera de las PC y verificar que la información no es recibida correctamente.

7º Configurar nuevamente los parámetros de las PC y repetir desde el 3er punto.

Como se mencionó al final del capítulo anterior, para el módulo de recepción asíncrona del puerto en el sistema PC-UNIX se contemplan 3 opciones: alarmas, señales y polling. Los resultados de cada método fueron los siguientes.

1.- Alarmas. La implementación por alarmas consiste en activar un cronómetro (alarma) que a su vez activará una función de software una vez expirado el tiempo del cronómetro.

La información que llega al puerto serie es almacenada en un bufer de 256 caracteres reservado por el archivo descriptor para este propósito.

La función que es llamada por la alarma verifica la existencia de datos en este bufer.

La resolución de la alarma° (tiempo mínimo) es de 1 segundo. Esta resolución impide de cierta manera algunas de las combinaciones de configuración (Bauds, Datos, Paridad y Paro) para poder recibir información de más de 256 caracteres.

Esto implica que al momento de configurar el puerto, debe tomarse en cuenta el tamaño del bufer, ya que de lo contrario se corre el riesgo de perder información por sobrescritura. Por ejemplo, si se configura el puerto con los siguientes parámetros:

Bauds 9600
Bits de dato 8
Paridad ..... PAR
Bits de paro 2

y se considera el caso más crítico, en el cual estamos recibiendo información constante durante un segundo, resultaría lo siguiente:

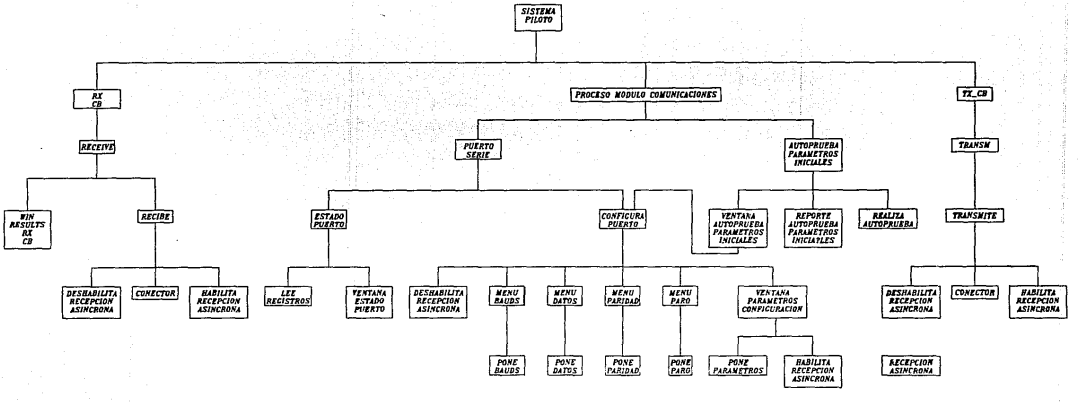

Figura III.3 Sistema UNIX para pruebas.

Se tiene una velocidad de transmisión de 9600 bauds[3], esto es, se tienen 9600 cambios de nivel en 1 segundo. Estos cambios de nivel son el total de: bits de inicio, bits de dato, bit de paridad y bits de paro. Para el caso expuesto se tiene un total de 12 bits por caracter de información enviado. Durante 1 segundo a la velocidad de 9600 bauds se tendrían un total de 9600/12 = 800 caracteres de información en 1 segundo.

Si el bufer sólo tiene capacidad para 256 caracteres, esto implica que únicamente podrán recuperar los últimos 256 caracteres, debido a que los primeros se perderán por sobrescritura. Para resolver este problema debería tenerse un bufer de 800 caracteres mínimo, lo cual no es posible puesto que la longitud la impone el archivo descriptor y éste a su vez es controlado por especificaciones del sistema operativo.

2.- Señales y Polling°. Para el caso de la implementación de recepción asíncrona mediante señales y del polling°, la situación no cambió en forma apreciable. En ambos casos se requiere configurar una terminal. Esta terminal es en realidad nuestro dispositivo externo, dicha terminal consume espacio en disco para su definición, además de tener ciertos procesos que son generados para el control de cualquier terminal en los sistemas UNIX. Otro inconveniente que se tiene con los sistemas de señales y polling, es que sólo es posible cuando el operador es superusuario[4]. Esto pone en riesgos al sistema cuando el usuario no tiene conocimiento amplio del mismo y por lo tanto pudiera causar daños al propio sistema.

Para el caso de las alarmas, el software funciona correctamente sin ser superusuario, sin embargo, tanto para las alarmas como para las señales y el polling el sistema tendría los requerimientos mínimos del puerto serie, que son: configuración,

[3] Ver capítulo I sección I.1.

[4] Un superusuario es el administrador del sistema, teniendo todos los privilegios y responsabilidad sobre éste. Estos privilegios incluyen permisos sobre todos los usuarios, así como de todos los procesos del sistema.

transmisión y recepción, sin contar con la autoprueba al CI 8250 ni el proceso de monitoreo del puerto y verificación de conector debido a que no se tiene acceso al canal de entrada/salida por estas vías.

Existe, sin embargo, una herramienta dentro del Venix que permite tener acceso al canal de entrada/salida. Esta herramienta son las rutinas de tiempo real[4] que permiten las opciones de autoprueba y monitoreo del estado del puerto aunque con la restricción de ser manejadas sólo con los permisos de un superusuario, ademas de limitar un poco la portabilidad de estas funciones a otros sistemas UNIX debido a que no son funciones estándar.

Por esto, las únicas rutinas que se encuentran condenadas a ser modificadas cuando se cambie de versión de sistema UNIX son: deshabilita recepción asíncrona, habilita recepción asíncrona, recepción asíncrona, realiza autoprueba, conector y lee registros.

No por ello, las rutinas de tiempo real dejan de ser una herramienta bastante poderosa debido a que cuentan con las siguientes ventajas sobre los sistemas convencionales de UNIX:

a) Permiten programar prioridades: El sistema en tiempo real permite asignar cierta prioridad a los procesos atendidos por el microprocesador. Esta característica es ampliamente usada para el módulo de recepción asíncrona debido a que ésta debe tener una prioridad alta en el módulo de comunicaciones en comparación con otras funciones o procesos dentro del sistema piloto.

b) Minimizar el tiempo latente de una interrupción: El tiempo latente de una interrupción se refiere a la cantidad de tiempo que le toma al sistema para reconocer una interrupción y para que se comience a atender. El sistema en tiempo real permite minimizar este tiempo de latencia de una interrupción. El módulo de recepción asíncrona es manejado por la interrupción por hardware número 3 para el puerto serie 2.

c) Entrada/Salida Directas: El sistema en tiempo real proporciona la facilidad de tener una confiable entrada/salida directa sin pasar por un bufer intermedio. En el caso del acceso a los registros del CI 8250, es la herramienta que se buscaba debido a que permite, sin depender de un almacenamiento secundario, leer o escribir información a los registros directamente. Con esta característica que proporciona el sistema en tiempo real se logró la codificación de los módulos para autopueba y verificación del conector.

Los sistemas en tiempo real deben satisfacer un mayor número de características que las descritas aquí, sin embargo, estas son las más relevantes para la implementación del módulo de comunicaciones.

Las funciones en tiempo real utilizadas son: *rtioset*, *rtpriority*, y el *rttrapint*[6].

rtioset: Esta función tiene la siguiente sintaxis:

```
int rtioset( int )
```

Esta función es utilizada para poder abrir el canal de entrada-salida y accesar el CI 8250. El parámetro pasado a está es un entero que especifica la acción que realizará la función. Esta acción para el caso del canal de entrada-salida tiene la mascara de RT_IO_ENABLE.

El valor regresado por esta función es un entero. Si dicho entero es mayor que cero entonces el proceso de apertura del canal de entrada-salida fue llevado exitosamente; si fue menor que cero, entonces existio un error al tratar de abrir el canal.

rtpriority: Esta función tiene la siguiente sintaxis:

```
int rtpriority( int , int )
```

Dicha función nos permite poner una prioridad al módulo de comunicaciones o bien quitar dicha prioridad dentro del sistema operativo. Si no se considera esta función, entonces podrán encontrarse retardos dentro del sistema que impiden tener una repuesta en tiempo real.

El primer parámetro es un entero que indica si la operación a realizar es la de poner (RT_PRI_SET) o quitar (RT_PRI_GET) la prioridad dentro del sistema operativo.

El segundo parámetro es un entero que permite identificar el nivel de prioridad que tendrá el módulo de comunicaciones dentro del sistema. Esta prioridad es la más baja (RT_PRI_LOW) dentro del sistema operativo.

El valor retornado por esta función es un entero el cual, si es mayor o igual a cero indica la prioridad del proceso dentro del sistema; si es menor a cero, entonces la colocación de la prioridad no fue posible.

rttrapint: Esta función tiene la siguiente sintaxis:

```
int rttrapint( int , int , int )
```

Esta función es requerida para activar la interrupción por hardware del puerto serie para el módulo de recepción asíncrona.

Los parámetros pasados a esta función, así como el valor retornado por la misma son todos entero. El primer parámetro se refiere a la acción de colocar (RT_IV_GET) o quitar (RT_IV_REL) la interrupción.

El segundo parámetro se refiere a la interrupción por hardware que será incluida en el vector de interrupciones del sistema operativo. Para el caso del módulo de comunicaciones es la interrupción 3 del puerto serie COM2.

Cuando se genera alguna interrupción en el sistema UNIX, ésta puede ser reconocida mediante una señal específica que envía el sistema operativo a todos aquellos procesos que la requieran. El tercer parámetro se refiere precisamente a esta señal que será generada por el sistema operativo, y que puede ser reconocida por software, cuando exista la interrupción por hardware. Esta señal es SIGUSR1 debido a que es requerida por un proceso de usuario.

Así como de las funciones *inb* y *outb*.

inb: Esta función tiene la siguiente sintaxis:

```
int inb( int )
```

Esta función es requerida para recibir la información de los registros del CI 8250.

El parámetro pasado a ésta, es la dirección de los registros del 8250. Para el módulo de comunicaciones, el valor regresado por ésta es el número ASCII del byte leído de los registros del 8250.

outb: Esta función tiene la siguiente sintaxis:

```
outb(M_C_REG_COM2,(unchar) 0x0A);
```

Esta función es requerida para enviar la información a los registros del CI 8250.

El primer parámetro pasado a ésta, es la dirección de los registros del 8250. El segundo parámetro es el valor regresado por ésta es el número ASCII del byte escrito a los registros del 8250.

Para el resto de las rutinas se usaron funciones estándares del lenguaje C como son: *ioctl, read, open* y el *write*[9].

Para entender la importancia de estas funciones es necesario que se tenga en mente que los dispositivos son manejados como archivos.Estos dispositivos pueden ser controlados mediante un programa o bien con comandos desde el *shell* (o interprete de comandos).

Desde un programa estos archivos pueden ser accesados con dos tipos de funciones: STREAMS o bien con LLAMADAS AL SISTEMA. Las primeras se generan con las funciones de biblioteca estándar de UNIX que son fopen, fclose, fwrite y fwread; sin embargo estas tienen un inconveniente debido a que generan un bufer intermedio entre el programa y el control del dispositivo por lo que son más lentas cuando se maneja gran cantidad de información. Las segundas se generan con las funciones open, close, write y read, las cuales hacen un llamado al sistema operativo para accesar directamente al controlador del dispositivo sin necesidad de que la información pase por un bufer intermedio.

Estas llamadas al sistema fueron utilizadas de la siguiente manera:

open: Esta función tiene la siguiente sintaxis:

int fd = open(char *path ,int oflag)

El primer parámetro (path) es la ruta y el nombre del archivo descriptor. Para el puerto serie 2 path es "/dev/tty01". Los archivos descriptores para el caso del puerto serie no pueden crearse únicamente abrirse.

El segundo parámetro (oflag) es la modo en que será trabajado el archivo descriptor. El modo elegido fue de lectura y escritura "O_RDWR".

El valor regresado por esta función (fd) es un entero secuencial que pertenece al archivo descriptor. Este entero debe ser mayor que 1.

read y *write*: Estas funciones tienen la siguiente sintaxis y fueron empleadas para la recepción y transmisión de información respectivamente:

```
int read( int fd , void *rx , unsigned nbytes )

int write( int fd , void *tx , unsigned nbytes ).
```

El primer parámetro (fd) es el archivo descriptor.

El segundo parámetro (rx o tx) es una variable caracter. Esta variable contiene la información que se recibe por el puerto (en el caso del read) o bien la información que será enviada por el puerto (en el caso del write).

El tercer parámetro (nbytes) indica el número de bytes que serán recibidos (read) o bien el número de bytes que serán transmitidos (write).

El valor retornado de esta funcion es un entero que indica el número de bytes recibidos o transmitidos.

ioctl: Esta función tiene la siguiente sintaxis y es empleada para configurar el puerto serie:

```
int ioctl( int fd , int request , /* arg */ )
```

El primer parámetro se refiere al archivo descriptor.

El segundo parámetro (request) se refiere a una de las petición que se pueden hacer para el puerto serie. El tercer parámetro (arg) depende de la petición realizada. Las peticiones son:

PETICION	ARG
TCGETA	ESTRUCTURA
TCSETA	ESTRUCTURA
FIORDCHK	NULO
TIOCFLUSH	NULO

La petición TCGETA obtiene los parámetros del puerto serie y los almacena en una ESTRUCTURA. La petición TCSETA pone los parámetros que contiene la ESTRUCTURA en el puerto serie. La petición FIORDCHK verifica en el archivo descriptor si no existe algún dato en el puerto para ser recibido. La última petición TIOCFLUSH limpia el puerto serie de cualquier información contenida en él.

III.3 PRUEBAS ENTRE PC-UNIX CON UN DISPOSITIVO EXTERNO.

Las pruebas realizadas entre PC-UNIX y el dispositivo externo consistieron en comunicarse con un dispositivo desarrollado en el Instituto de Investigaciones Eléctricas.

Este dispositivo, denominado miniUTR, recibe una U o una M como caracter de control e inmediatamente envía información según el caracter recibido como se muestra en la siguiente tabla.

CARACTER RECIBIDO	NUMERO DE BYTES ENVIADOS
U	4
M	VARIABLE

Cuando se envía una U a la miniUTR ésta envía un total de 4 bytes de información y cuando se envía una M ésta envía lo que tenga en una memoria RAM que utiliza para almacenar cierta información.

Las pruebas aplicadas entre el sistema PC-UNIX y la miniUTR consistieron en:

1° Configurar el sistema PC-UNIX con los parámetros de operación de la miniUTR. Estos parámetros son: 300 Bauds, 8 Bits de dato, Sin paridad y 1 Bit de paro.

2° Enviar cualquier caracteres de control (U o M) a la miniUTR.

3° Recibir la información con la rutina de recepción asíncrona.

4° Desplegar la información recibida una vez terminada la recepción de información.

Las primeras pruebas realizadas entre estos sistemas indicaron perdida de información por sobrescritura bloqueando completamente la rutina de recepción asíncrona.

El error por el cual se generó esta situación fue el no verificar el estado del registro de interrupciones. Este registro indica cuando en el puerto serie, y específicamente en el CI 8250, se ha generado alguna situación (dato recibido, error

en la línea, registro de transmisión vacío) que activa la señal de interrupción del puerto serie . Si el registro no es leído, entonces la señal generada por éste es capturada por el sistema operativo UNIX el cual bloquea cualquier operación realizada sobre el puerto serie.

Una vez leído dicho registro, el sistema UNIX sobrentiende que el usuario se ha percatado de lo que ocurre en el puerto serie y restablece la función de recepción asíncrona permitiendo la operación normal del sistema.

Con este último punto quedo concluida la validación de las pruebas del módulo de comunicaciones. Con este módulo de comunicaciones es posible implementar algún protocolo de comunicaciones mediante las rutinas de transmisión y recepción por peticiones de usuario.

CONCLUSIONES

El desarrollo de un sistema que permita controlar el puerto de comunicaciones fue el objetivo de este trabajo de tesis, el cual contempla varios puntos:

1.- Permitir transmisión y recepción de información proveniente de cualquier dispositivo externo que maneja RS-232.

2.- Configurar al puerto con los parámetros de operación requeridos.

3.- Permitir verificar el estado en que se encuentra el puerto serie.

4.- Verificar la existencia de un conector como parámetro para determinar la presencia del medio de comunicación.

5.- Ser modular para permitir su rápida incorporación a cualquier programa que lo requiera.

Para cumplir con el objetivo se tuvo que desarrollar un software, que por medio de menúes y ventanas gráficas cumpliera con los puntos estipulados por éste. Dicho software debía trabajar en el ambiente del sistema operativo UNIX.

Para implementar el software fue necesario determinar los recursos con los que se contaba, tanto de software como de hardware. Estos recursos determinarían la ruta a tomar para el desarrollo del software.

Para desarrollar el sistema se contemplaron diversas alternativas de programación que presta el sistema UNIX, sin embargo cada una de éstas tiene sus ventajas y desventajas; sin embargo, estas opciones presentaron una desventaja en común debido a la filosofía que maneja el sistema UNIX para con el manejo de los periféricos a éste conectados: LIMITAR AL USUARIO EL CONTROL DEL PERIFERICO.

Con esta directiva, se generó dicho software de manera modular que permite su fácil incorporación con una simple instrucción de Include en lenguaje C, mientras que los menús y ventanas gráficas fueron implementadas con XWINDOW. Para el cumplimiento de la configuración, el envío y recepción de información se dispone de las rutinas en lenguaje C que llevan a cabo de manera directa estos procesos; dichas funciones son: ioctl, read y write. Sin embargo, para el caso de la recepción asíncrona, monitoreo y verificación del conector no existen funciones que realicen de manera directa estas operaciones.

Para la implementación de la función de recepción asíncrona, se probaron 3 posibles caminos:

1º Alarmas.
2º Señales.
3º Polling.

La primera opción enfrenta el problema de tener un tiempo mínimo para activar una función, la cual verifica la existencia de datos en el puerto serie, de 1 segundo; lo que crea problemas cuando se tiene una configuración de 4800 Bauds, 8 bits de dato, paridad y dos bits de paro por ejemplo. Este tiempo mínimo de 1 segundo provoca la pérdida de información en algunas combinaciones de la configuración por sobreescritura del bufer de recepción.

La segunda y tercera opción se encuentran sujetas al sistema operativo. Esto es debido a que es necesario configurar dentro del sistema operativo una terminal, que estará supuestamente conectada al puerto serie, para poder hacer uso de estas opciones.

Los tres caminos resuelven el problema de la recepción asíncrona, sin embargo, el problema del monitoreo y la verificación del conector no es resuelto por estos métodos.

Las rutinas de tiempo real no presentaron los problemas encontrados con las alarmas, las señales o el polling, permitiendo a su vez el monitoreo del puerto serie así como la verificación del conector. Sin embargo, presentan las desventajas de no ser portable y requerir ser superusuario para el manejo de las rutinas con lo que el sistema queda vulnerable a cualquier manejo indebido de los recursos proporcionados a estos superusuarios.

De las pruebas realizadas al módulo de comunicaciones, una vez que se tomó el camino a seguir, resultaron los siguientes sistemas:

1° Un sistema PC-DOS con los puntos principales del objetivo de la tesis: Transmisión, recepción, configuración, autoprueba, monitoreo y manejo de menúes. Este sistema trabaja en base al sistema operativo DOS y fue generado como punto de partida para la validación del software de comunicaciones.

2° Una versión del módulo de comunicaciones que permite observar, en ventanas gráficas, la información enviada o recibida con la finalidad de monitorear lo que está sucediendo durante las pruebas entre PC-UNIX y PC-DOS o un dispositivo externo.

3° El sistema final, módulo de comunicaciones, que ya no incluye las ventanas de monitoreo del proceso de comunicaciones.

Con estos sistemas se logró probar y validar el software generado para el sistema de comunicaciones que, trabajando bajo el sistema UNIX y con el uso de rutinas en tiempo real, cumple con los requerimientos para el caso específico del sistema piloto. Sin embargo, para trabajos futuros es recomendable eliminar la parte de tiempo real para permitir portabilidad e independencia de la versión del sistema UNIX, así como la sustitución de las funciones open, read, write e ioctl que no se encuentran dentro del estándar ANSI[1]. Para lograr ésto, es necesario crear un device driver que proporcione el mismo control del puerto serie pero interactuando directamente con los procesos de control del sistema operativo.

[1] El estándar ANSI para el sistema operativo UNIX no contempla estas funciones y recomienda el uso de STREAMS para el manejo y control de archivos descriptores.

```
/*************************************************************

     Archivo cabecera:

                COM.H

     Este archivo contiene las funciones para el manejo del puerto serie
     de comunicaciones

     Los parámeros para COM2 son:

     Bauds ..... 2400
     Paridad ..... Ninguna
     Bitsa de paro ..... 1
     Bits de da dato ... 8
     Modo ..... recepción

*************************************************************/


/*************************************************************
     Archivos cabecera.
*************************************************************/
#include <Xm/BulletinB.h>
#include <Xm/Label.h>
#include <Xm/FileSB.h>
#include <Xm/Text.h>
#include <Xm/SelectioB.h>
#include <Xm/MessageB.h>
#include <X11/Shell.h>
#include <string.h>
#include <limits.h>
#include <sys/inline.h>
#include <sys/stat.h>
#include <sys/signal.h>
#include <stdio.h>
```

```
#include <termio.h>
#include <unistd.h>
#include <errno.h>
#include <rtx.h>


/*************************************************************
     Constantes globales.
*************************************************************/
#define COM2            "/dev/tty01"
#define INT_COM2        3          /* Interrupt to COM2 */
#define RX_REG_COM2     0x2F8        /* Receiver Data Register in COM2 */
#define TX_REG_COM2     0x2F8      /* Transmitter Holding Register in COM2 */
#define D_L_L_COM2      0x2F8        /* Divisor Latch Least in COM2 */
#define D_L_M_COM2      0x2F9        /* Divisor Latch Most in COM2 */
#define I_E_REG_COM2    0x2F9      /* Interrupt Enable Register in COM2 */
#define L_C_REG_COM2    0x2FB        /* Line Control Register in COM2 */
#define M_C_REG_COM2    0x2FC        /* Modem Control Register in COM2 */
#define L_S_REG_COM2    0x2FD        /* Line Status Register in COM2 */
#define M_S_REG_COM2    0x2FE        /* Modem Status Register in COM2 */
#define M_C_LOOP_ON     0x10       /* Value in Modem Control for ON Loopback
                        */
#define M_C_LOOP_OFF    0xEF       /* Value in Modem Control for OFF Loopback
                        */
#define ON              1
#define OFF             0
#define YES               1
#define RX_BYTE           1
#define RX_STRING         2
#define RX_FILE           3
#define PATH_RX_FILE    "/usr/rick/basuras/rx_file.dat"
#define TX_BYTE           1
#define TX_STRING         2
#define TX_FILE           3
#define DB_7              CS7
#define DB_8              CS8
#define PAR_ODD           ( PARENB | PARODD )
```

```
#define PAR_EVEN        PARENB
#define NO_PAR          0
#define STP_1           0
#define STP_2           CSTOPB
#define ALARM           4
#define EOF_RX          0x04    /* EOT or EOF in UNIX */
#define EOS_RX          0x00    /* End Of String in RX*/
#define EOB_RX          0x00    /* End Of Byte in RX */
#define EOF_TX          0x1A    /* SUB or EOF in DOS */
#define EOS_TX          0x00    /* End Of String in TX */
#define EOB_TX          0x00    /* End Of Byte in TX */
#define CHAR_MAX_ERR -1     /* OverFlow Error in CHAR_MAX */
#define CONN_PRES       0x0002  /* Conector is present */
#define CONN_ERR        0x0003  /* Conector not set */
#define RX_DATA_RDY  0x0001  /* Receiver Data Ready */
#define DATA_ERR        0x0001  /* Data error in transmission */
#define OVER_ERR     0x0002  /* Overrun Error in Rx */
#define PAR_ERR      0x0004  /* Parity Error in Rx */
#define FRAM_ERR     0x0008  /* Framing error */
#define BREAK_ERR       0x0010  /* All frame is zeroes */
#define TX_RDY       0x0040  /* Enable to TX */
#define ENBL_IO_ERR     -1      /* Error in rtioset function */
#define SET_PRI_ERR     -2      /* Set priority error */
#define GET_INT_ERR     -3      /* Get interrupt vector error */
#define REL_INT_ERR     -4      /* Release interrupt vector error */
#define RX_LOOP_ERR     -2      /* Bad RX in loopback */
#define OPEN_FILDES_ERR -3  /* Can't to open fildes */
#define PAR_FILDES_ERR -4       /* Can't set parameters in fildes */


/*********************************************************
      Variables globales.
*********************************************************/
char  buff[CHAR_MAX],buff_rx[CHAR_MAX];
char  buff_tx[CHAR_MAX],buff_rx_int[1500];
struct termio CONFIGURACION;
int    fildes=0,ENBL_COM=ON,DISPLAY=ON,count_rx=0;
```

```
int    CONF_LOOPBACK = OFF;
static Widget wid_loopback,Port_button,Port_parent;


/*************************************************************
        Variables para las ventans.
*************************************************************/
unsigned char str_bauds[9] = " 2400\n",str_parity[12] = " Ninguna\n";
unsigned char str_stop[6] = " 1\n",str_data[6] = " 8\n",str_com[6] = " 2\n";


/*************************************************************
        Funciones prototipo.
*************************************************************/
void  Destroy_CB();
void  Destroy_conf_CB();
void  rx_CB();
void  serial_port_CB();
void  tx_CB();
void  win_results_Rx_CB();
void  estado_puerto();
void  configura_puerto();
void  menue_bauds();
void  menue_dato();
void  menue_paridad();
void  menue_paro();
void  ventana_parametros_configuracion();
void  pone_bauds();
void  pone_dato();
void  pone_paridad();
void  pone_paro();
void  receive();
void  win_tx_CB();
void  recepcion_asincrona();
void  Print_rx_asinc();
```

```
/**************************************************************
      Estructuras para las ventanas y menúes.
**************************************************************/
typedef struct bull_dialog_struct
{
 char              *label_button; /* Etiqueta para el boton */
 void              (*func)();    /* Funcion a llamar */
 Position           pos_x;       /* Posicion del boton en x */
 Position           pos_y;       /* Posicion del boton en y */
 Position           offset_x;    /* Desplazamiento en x del padre */
 Position           offset_y;    /* Desplazamiento en y del padre */
 Int               parameter;    /* Parametro a pasar  */
} Bulletin_struct;

static Bulletin_struct Estructura_modulo_transmite[] = {
 {" Caracter ",win_tx_CB,0,0,164,34,TX_BYTE},
 {" Cadena ",win_tx_CB,82,0,0,0,TX_STRING},
 {" Archivo ",win_tx_CB,147,0,0,0,TX_FILE},
 {" Salir ",Destroy_CB,212,0,0,0,0}
};

static Bulletin_struct Estructura_modulo_Comunicaciones[] = {
 {" Recibe ",rx_CB,0,0,-100,54,0},
 {" Puerto Serie ",serial_port_CB,70,0,0,0,0},
 {" Transmite ",tx_CB,165,0,0,0,0},
 {" Rx Asinc ",Print_rx_asinc,245,0,0,0,0},
 {" Salir ",Destroy_CB,323,0,0,0,0}
};

static Bulletin_struct Estructura_modulo_recibe[] = {
 {" Caracter ",receive,0,0,9,34,RX_BYTE},
 {" Cadena ",receive,82,0,0,0,RX_STRING},
 {" Archivo ",receive,147,0,0,0,RX_FILE},
 {" Salir ",Destroy_CB,212,0,0,0,0}
};
```

```
static Bulletin_struct Estructura_modulo_puerto_serie[] = {
 {" Configura ",configura_puerto,0,0,70,34,0},
 {" Monitoreo ",estado_puerto,88,0,0,0,0},
 {" Salir ",Destroy_CB,170,0,0,0,0}
};

static Bulletin_struct Estructura_configura_estado[] = {
 {" Configura ",configura_puerto,0,0,70,34,0},
 {" Monitoreo ",estado_puerto,88,0,0,0,0},
 {" Salir ",Destroy_CB,170,0,0,0,0}
};

static Bulletin_struct Estructura_modulo_configura[] = {
 {" Bauds     .",menue_bauds,0,0,0,34,0},
 {" Datos     .",menue_dato,0,35,0,0,0},
 {" Paridad   .",menue_paridad,0,60,0,0,0},
 {" Paro      .",menue_paro,0,85,0,0,0},
 {" Confirma  .",ventana_parametros_configuracion,0,110,0,0,0},
 {" Salir     .",Destroy_conf_CB,0,135,0,0,0}
};

static Bulletin_struct Estructura_menue_bauds[] = {
 {" 110  ",pone_bauds,0,0,83,9,B110},
 {" 150  ",pone_bauds,0,35,0,0,B150},
 {" 300  ",pone_bauds,0,60,0,0,B300},
 {" 600  ",pone_bauds,0,85,0,0,B600},
 {" 1200 ",pone_bauds,0,110,0,0,B1200},
 {" 2400 ",pone_bauds,0,135,0,0,B2400},
 {" 4800 ",pone_bauds,0,160,0,0,B4800},
 {" 9600 ",pone_bauds,0,185,0,0,B9600}
};

static Bulletin_struct Estructura_menue_datos[] = {
 {" 7 bits de dato ",pone_dato,0,0,83,35,DB_7},
 {" 8 bits de dato ",pone_dato,0,35,0,0,DB_8}
};
```

```
static Bulletin_struct Estructura_menue_paridad[] = {
 {" Par     ",pone_paridad,0,0,83,59,PAR_EVEN},
 {" Impar   ",pone_paridad,0,35,0,0,PAR_ODD},
 {" Ninguna ",pone_paridad,0,60,0,0,NO_PAR}
};

static Bulletin_struct Estructura_menue_paro[] = {
 {" 1 bit de paro  ",pone_paro,0,0,83,85,STP_1},
 {" 2 bits de paro ",pone_paro,0,35,0,0,STP_2}
};


/*************************************************************
      Inicia código principal. El código muestra tambien las funciones utilizadas
      para las pruebas.
*************************************************************/
void Print_rx_asinc(w,parent,call_data)
Widget    w;
Widget    parent;
caddr_t   call_data;
{
int i;
for(i=0;i<count_rx;i++)
printf(" buff_rx_int[%d] = %x,",i,(unsigned int)buff_rx_int[i]);
printf("\n");
printf("%s   ",buff_rx_int);
printf("%d\n",count_rx);
}


int deshabilita_recepcion_asincrona()
{
sigset(SIGUSR1,SIG_IGN);
if(rttrapint(RT_IV_REL,INT_COM2,SIGUSR1) == -1) return REL_INT_ERR;
rtpriority(RT_PRI_SET,RT_PRI_OFF);
return 0;
}
```

```
int habilita_recepcion_asincrona()
{
if(rtioset(RT_IO_ENABLE)<0) return ENBL_IO_ERR;
if(rtpriority(RT_PRI_SET,RT_PRI_HIGH)<0) return SET_PRI_ERR;
sigset(SIGUSR1,recepcion_asincrona);
if(rttrapint(RT_IV_GET,INT_COM2,SIGUSR1)==-1) return GET_INT_ERR;
return 0;
}

void recepcion_asincrona()
{
inb(0x2F9);
inb(0x2FA);
if(!(inb(L_S_REG_COM2)&RX_DATA_RDY))
while(!(inb(L_S_REG_COM2)&RX_DATA_RDY)&&i<100000) i++;
buff_rx_int[count_rx] = rtio_inb(RX_REG_COM2);
count_rx++;
sigset(SIGUSR1,recepcion_asincrona);
}

int conector()
{
outb(M_C_REG_COM2,(unchar) 0x0A);
inb(M_S_REG_COM2);
outb(M_C_REG_COM2,(unchar) 0x0B);
if(inb(M_S_REG_COM2)==0) return CONN_ERR;
else return CONN_PRES;
}

void Destroy_CB(w,parent,call_data)
Widget    w;
Widget    parent;
caddr_t   call_data;
{
 XtSetSensitive(parent,TRUE);
 XtDestroyWidget(XtParent(w));
```

```
void Destroy_conf_CB(w,parent,call_data)
Widget    w;
int       parent;
caddr_t   call_data;
{
   XtSetSensitive(XtParent(XtParent(XtParent(w))),TRUE);
   XtDestroyWidget(XtParent(XtParent(w)));
 If(CONF_LOOPBACK = =ON){
    XtDestroyWidget(XtParent(wid_loopback));
    CONF_LOOPBACK = OFF;
  }
habilita_recepcion_asincrona();
}

static Widget CreateBulletinBoard(title,parent,bull_struct,n,parent_x,parent_y)
char              *title;
Widget            parent;
Bulletin_struct   *bull_struct;
int               n;
Position          parent_x;
Position          parent_y;
{
static Widget     Bull_Board;
int               i=0;
WidgetList        buttons;
Arg               args[6];

buttons = (WidgetList) XtMalloc(n * sizeof(Widget));

XtSetArg(args[0],XmNdefaultPosition,FALSE);
XtSetArg(args[1],XmNautoUnmanage,FALSE);
XtSetArg(args[2],XmNnoResize,TRUE);
XtSetArg(args[3],XmNx,(Position) bull_struct[0].offset_x+parent_x);
XtSetArg(args[4],XmNy,(Position) bull_struct[0].offset_y+parent_y);
XtSetArg(args[5],XtNtitle,title);
Bull_Board = XmCreateBulletinBoardDialog(parent,NULL,args,XtNumber(args));
```

```
for(i=0;i<n;i++){
 buttons[i] = XtCreateManagedWidget(bull_struct[i].label_button,
          xmPushButtonWidgetClass,Bull_Board,args,XtNumber(args));
        if( bull_struct[i].func==Destroy_CB)
                  XtAddCallback(buttons[i],XmNactivateCallback,
                  bull_struct[i].func,parent);
        else XtAddCallback(buttons[i],XmNactivateCallback,
                  bull_struct[i].func,bull_struct[i].parameter);
        XtMoveWidget(buttons[i],bull_struct[i].pos_x,bull_struct[i].pos_y);
  }
return Bull_Board;
}

void win_results_Rx_CB(w,str_id_rx,x,y)
Widget     w;
char            *str_id_rx;
Position        x;
Position        y;
{
static Widget      Dialog_Rx;
Arg             args[6];
Position        x_p=0,y_p=0;
XtSetSensitive(w,FALSE);
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(XtParent(w),args,2);
XtSetArg(args[0],XmNnoResize,TRUE);
XtSetArg(args[1],XmNautoUnmanage,FALSE);
XtSetArg(args[2],XmNmessageString,XmStringCreateLtoR(str_id_rx,
      XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[3],XmNokLabelString,XmStringCreateLtoR(" Salir ",
      XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[4],XmNdefaultPosition,FALSE);
XtSetArg(args[5],XtNtitle," RECEPCION ");
Dialog_Rx=XmCreateInformationDialog(XtParent(w),NULL,args,XtNumber(args));
XtUnmanageChild(XmMessageBoxGetChild(Dialog_Rx,
          XmDIALOG_HELP_BUTTON));
```

```
XtUnmanageChild(XmMessageBoxGetChild(Dialog_Rx,
            XmDIALOG_CANCEL_BUTTON));
XtAddCallback(Dialog_Rx,XmNokCallback,Destroy_CB,w);
XtMoveWidget(Dialog_Rx,x_p+x,y_p+y);
XtManageChild(Dialog_Rx);
}

int recibe(ID_RX)
int ID_RX;
{
int  num_char  =  0,tot_char=0,conn=0;
FILE *fd;
deshabilita_recepcion_asincrona();
conn=conector();
if(conn==CONN_ERR){
   habilita_recepcion_asincrona();
   return CONN_ERR; /* Error por no haber conector */
 }
memset(buff_rx,(int) ' ',sizeof(buff_rx));
buff_rx[CHAR_MAX-1] = '\0';
while(1){
      if(ioctl(fildes,FIORDCHK,NULL)>0){
            while((num_char = read(fildes,buff,CHAR_MAX))==0);
            tot_char=num_char;
            switch(ID_RX){
                  case RX_BYTE:
                        strncpy(buff_rx,buff,1);
                        buff_rx[1]='\0';
                        while(read(fildes,buff,CHAR_MAX)>0);
                        ioctl(fildes,TIOCFLUSH,NULL);
                        habilita_recepcion_asincrona();
                        break;
                  case RX_STRING:
                       strncpy(buff_rx,buff,num_char);
                       buff_rx[tot_char) = '\0';
                         if(buff_rx[tot_char-1]==(char) EOS_RX){
                               habilita_recepcion_asincrona();
```

```
                return 0;
            }
        while(1){
          while((num_char=read(fildes,buff,CHAR_MAX))==0);
          tot_char = num_char + tot_char;
          strncat(buff_rx,buff,num_char);
            buff_rx[tot_char] = '\0';
            if(buff_rx[tot_char-1]==(char) EOS_RX){
                habilita_recepcion_asincrona();
                return 0;
              }
          else if(tot_char>=CHAR_MAX){
                habilita_recepcion_asincrona();
                return CHAR_MAX_ERR;
              }
          }
          break;
      case RX_FILE:
          fd = fopen(PATH_RX_FILE,"w+");
          strncpy(buff_rx,buff,num_char);
          buff_rx[num_char]='\0';
          if(strchr(buff_rx,EOF_RX)!=NULL){
            fwrite(buff_rx,1,tot_char,fd);
            fclose(fd);
            habilita_recepcion_asincrona();
            return 0;
           }
          while(1){
      while((num_char=read(fildes,buff,CHAR_MAX))==0);
           tot_char = num_char + tot_char;
           strncat(buff_rx,buff,num_char);
            buff_rx[tot_char] = '\0';
           if(strchr(buff_rx,EOF_RX)!=NULL){
              fwrite(buff_rx,1,tot_char,fd);
            fclose(fd);
               habilita_recepcion_asincrona();
              return 0;
```

```
                        }
                      else if(tot_char>=(CHAR_MAX-5)){ /* CHAR_MAX */
                        fwrite(buff_rx,1,tot_char,fd);
                        memset(buff_rx,(int) ' ',sizeof(buff_rx));
                        buff_rx[CHAR_MAX-1] = '\x0';
                        tot_char=0;
                        }
                      }
                    break;
            } /* Fin del switch */
        return 0;
       } /* Fin del if */
  } /* Fin del while */
}

void receive(w,ID_RX,call_data)
Widget    w;
int       ID_RX;
caddr_t   call_data;
{
int rx_error=0;
memset(buff_rx,(int) ' ',CHAR_MAX);
buff_rx[CHAR_MAX-1] = '\0';
switch (ID_RX){
  case RX_BYTE: /* Rx Byte */
            rx_error=recibe(ID_RX);
            if(DISPLAY==ON){
                  char str[40];
                  strcpy(str,"Byte: ");
                  if(rx_error!=0) strcat(str,"\n Error en recepcion\n");
                  else strcat(str,buff_rx);
                  win_results_Rx_CB(w,str,9,33);
                }
            break;
  case RX_STRING: /* Rx String */
            rx_error = recibe(ID_RX);
            if(DISPLAY==ON){
```

```
                    char str[CHAR_MAX];
                    strcpy(str,"Cadena: ");
                    if(rx_error! = 0) strcat(str,"\n Error en recepcion\n");
                    else strcat(str,buff_rx);
                    win_results_Rx_CB(w,str,82,33);
                  }
              break;
 case RX_FILE: /* Rx File */
              rx_error = recibe(ID_RX);
              if(DISPLAY = = ON){
                    char str[50];
                    strcpy(str,"Archivo: ");
                    if(rx_error! = 0) strcat(str,"\n Error en recepcion\n");
                  else strcat(str,PATH_RX_FILE);
                    win_results_Rx_CB(w,str,14,103);
                  }
              break;
 }
}

void rx_CB(w,client_data,call_data)
Widget  w;
int          client_data;
caddr_t      call_data;
{
static Widget   BullB;
Position        x_p,y_p;
Arg     args[2];
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(XtParent(w),args,2);
XtSetSensitive(w,FALSE);
BullB = CreateBulletinBoard(" RECIBE ",w,Estructura_modulo_recibe,
        XtNumber(Estructura_modulo_recibe),x_p,y_p);

XtManageChild(BullB);
}
```

```
void pone_parametros(w,parent,call_data)
Widget      w;
Widget      parent;
caddr_t             call_data;
{
ioctl(fildes,TCSETA,&CONFIGURACION);
XtDestroyWidget(w);
XtSetSensitive(parent,TRUE);
}

void ventana_parametros_configuracion(w,client_data,call_data)
Widget  w;
int         client_data;
caddr_t     call_data;
{
static Widget     Q_Dialog;
Arg               args[9];
Position          x_p=0,y_p=0;
static char       str_param[500];
XtSetSensitive(w,FALSE);
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(XtParent(w),args,2);
sprintf(str_param,"\
  Configuracion correcta con :\n\
\n\
* Velocidad en Bauds .....");
strcat(str_param,str_bauds);
strcat(str_param,"* Bits de dato .....");
strcat(str_param,str_data);
strcat(str_param,"* Bit de paridad .....");
strcat(str_param,str_parity);
strcat(str_param,"* Bits de paro .....");
strcat(str_param,str_stop);
strcat(str_param,"\0");
XtSetArg(args[0],XmNnoResize,TRUE);
XtSetArg(args[1],XmNautoUnmanage,FALSE);
```

```
XtSetArg(args[2],XmNmessageString,XmStringCreateLtoR(str_param,
        XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[3],XmNcancelLabelString,XmStringCreateLtoR(" No ",
        XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[4],XmNokLabelString,XmStringCreateLtoR(" Si ",
        XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[5],XmNdefaultPosition,FALSE);
XtSetArg(args[6],XmNx,x_p+77);
XtSetArg(args[7],XmNy,y_p+110);
XtSetArg(args[8],XtNtitle," CONFIRMA ");
Q_Dialog = XmCreateQuestionDialog(XtParent(w),NULL,args,XtNumber(args));
XtUnmanageChild(XmMessageBoxGetChild(Q_Dialog,XmDIALOG_HELP_BUTTON));
XtAddCallback(Q_Dialog,XmNcancelCallback,Destroy_CB,XtParent(w));
XtAddCallback(Q_Dialog,XmNokCallback,pone_parametros,w);
XtManageChild(Q_Dialog);
}

void pone_paro(w,ID_STOP,call_data)
Widget  w;
int          ID_STOP;
caddr_t      call_data;
{
ushort       filter = 0xFFBF; /* filter = 1111 1111 1011 1111 */
CONFIGURACION.c_cflag = CONFIGURACION.c_cflag & filter ;
switch(ID_STOP){
 case STP_1:
      strcpy(str_stop," 1\n");
      break;
 case STP_2:
       CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | STP_2 );
      strcpy(str_stop," 2\n");
      break;
 }
XtSetSensitive(XtParent(XtParent(XtParent(w))),TRUE);
XtUnmanageChild(XtParent(w));
}
```

```
void menue_paro(w,client_data,call_data)
Widget  w;
int          client_data;
caddr_t      call_data;
{
static Widget   BullB;
Position        x_p,y_p;
Arg             args[2];
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(XtParent(w),args,2);
XtSetSensitive(w,FALSE);
BullB = CreateBulletinBoard(" PARO ",w,Estructura_menue_paro,
        XtNumber(Estructura_menue_paro),x_p,y_p);
XtManageChild(BullB);
}

void pone_paridad(w,ID_PARITY,call_data)
Widget  w;
int     ID_PARITY;
caddr_t call_data;
{
ushort  filter = 0xFCBF; /* filter = 1111 1100 1111 1111 */
CONFIGURACION.c_cflag = CONFIGURACION.c_cflag & filter ;
switch(ID_PARITY){
 case PAR_EVEN:
        CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | PAR_EVEN );
       strcpy(str_parity," Par\n");
       break;
 case PAR_ODD:
        CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | PAR_ODD );
       strcpy(str_parity," Impar\n");
       break;
 case NO_PAR:
       strcpy(str_parity," Ninguna\n");
       break;
 }
```

```
XtSetSensitive(XtParent(XtParent(XtParent(w))),TRUE);
XtUnmanageChild(XtParent(w));
}

void menue_paridad(w,client_data,call_data)
Widget  w;
int     client_data;
caddr_t call_data;
{
static Widget   BullB;
Position x_p,y_p;
Arg     args[2];
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(XtParent(w),args,2);
XtSetSensitive(w,FALSE);
BullB = CreateBulletinBoard(" PARIDAD ",w,Estructura_menue_paridad,
        XtNumber(Estructura_menue_paridad),x_p,y_p);
XtManageChild(BullB);
}

void pone_dato(w,ID_DATA,call_data)
Widget  w;
int     ID_DATA;
caddr_t call_data;
{
ushort  filter = 0xFFCF; /* filter = 1111 1111 1100 1111 */
CONFIGURACION.c_cflag = CONFIGURACION.c_cflag & filter ;
switch(ID_DATA){
 case DB_7:
        CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | DB_7 );
       strcpy(str_data," 7\n");
       break;
 case DB_8:
        CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | DB_8 );
       strcpy(str_data," 8\n");
       break;
```

```
}
XtSetSensitive(XtParent(XtParent(XtParent(w))),TRUE);
XtUnmanageChild(XtParent(w));
}

void menue_dato(w,client_data,call_data)
Widget  w;
int     client_data;
caddr_t call_data;
{
static Widget   BullB;
Position x_p,y_p;
Arg     args[2];
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(XtParent(w),args,2);
XtSetSensitive(w,FALSE);
BullB = CreateBulletinBoard(" DATOS ",w,Estructura_menue_datos,
        XtNumber(Estructura_menue_datos),x_p,y_p);
XtManageChild(BullB);
}

void pone_bauds(w,ID_BAUDS,call_data)
Widget  w;
int     ID_BAUDS;
caddr_t call_data;
{
ushort  filter = 0xFFF0; /* filter = 1111 1111 1111 0000  */
CONFIGURACION.c_cflag = CONFIGURACION.c_cflag & filter ;
switch(ID_BAUDS){
 case B110:
       CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | B110 );
       strcpy(str_bauds," 110\n");
       break;
 case B150:
       CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | B150 );
       strcpy(str_bauds," 150\n");
```

```
        break;
case B300:
        CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | B300 );
        strcpy(str_bauds," 300\n");
        break;
case B600:
        CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | B600 );
        strcpy(str_bauds," 600\n");
        break;
case B1200:
        CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | B1200 );
        strcpy(str_bauds," 1200\n");
        break;
case B2400:
        CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | B2400 );
        strcpy(str_bauds," 2400\n");
        break;
case B4800:
        CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | B4800 );
        strcpy(str_bauds," 4800\n");
        break;
case B9600:
        CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | B9600 );
        strcpy(str_bauds," 9600\n");
        break;
}
XtSetSensitive(XtParent(XtParent(XtParent(w))),TRUE);
XtUnmanageChild(XtParent(w));
}

void menue_bauds(w,client_data,call_data)
Widget  w;
int     client_data;
caddr_t call_data;
{
static Widget   BullB;
Position x_p,y_p;
```

```
Arg     args[2];
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(XtParent(w),args,2);
XtSetSensitive(w,FALSE);
BullB = CreateBulletinBoard(" BAUDS ",w,Estructura_menue_bauds,
      XtNumber(Estructura_menue_bauds),x_p,y_p);
XtManageChild(BullB);
}

void configura_puerto(w,loopback,call_data)
Widget  w;
int     loopback;
caddr_t call_data;
{
static Widget   BullB;
Position x_p,y_p;
Arg     args[2];
if(loopback==YES){
      Widget wid_child;
      wid_child = XmMessageBoxGetChild(w,XmDIALOG_OK_BUTTON);
      XtSetArg(args[0],XmNx,&x_p);
      XtSetArg(args[1],XmNy,&y_p);
      XtGetValues(w,args,2);
      XtSetSensitive(wid_child,FALSE);
      wid_loopback = w;
      w=XtParent(w);
      CONF_LOOPBACK=ON;
  }
else {
      deshabilita_recepcion_asincrona();
      XtSetArg(args[0],XmNx,&x_p);
      XtSetArg(args[1],XmNy,&y_p);
      XtGetValues(XtParent(w),args,2);
      XtSetSensitive(w,FALSE);
   }
BullB = CreateBulletinBoard(" CONFIGURA",w,Estructura_modulo_configura,
```

```
        XtNumber(Estructura_modulo_configura),x_p,y_p);
XtManageChild(BullB);
}

void ventana_estado_puerto(w,report)
Widget w;
char  *report;
{
static Widget  Mess_Dialog;
Arg       args[8];
Position  x_p=0,y_p=0;
XtSetSensitive(w,FALSE);
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(XtParent(w),args,2);
XtSetArg(args[0],XmNnoResize,TRUE);
XtSetArg(args[1],XmNautoUnmanage,FALSE);
XtSetArg(args[2],XmNokLabelString,XmStringCreateLtoR(" Salir ",
      XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[3],XmNmessageString,XmStringCreateLtoR(report,
      XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[4],XmNdefaultPosition,FALSE);
XtSetArg(args[5],XmNx,x_p+90);
XtSetArg(args[6],XmNy,y_p+32);
XtSetArg(args[7],XtNtitle," ESTADO DEL PUERTO ");
Mess_Dialog=XmCreateMessageDialog(XtParent(w),NULL,args,XtNumber(args));
XtUnmanageChild(XmMessageBoxGetChild(Mess_Dialog,XmDIALOG_HELP_BUT
TON));
XtUnmanageChild(XmMessageBoxGetChild(Mess_Dialog,XmDIALOG_CANCEL_B
UTTON));
XtAddCallback(Mess_Dialog,XmNokCallback,Destroy_CB,w);
XtManageChild(Mess_Dialog);
}
```

```
void lee_registros(report)
char *report;
{
int  line_status=0,conn=0;
deshabilita_recepcion_asincrona();
line_status = inb(L_S_REG_COM2);
conn = conector();
habilita_recepcion_asincrona();
report[0]='\0';
if(conn!=CONN_PRES) strcat(report," No existe conector\n");
if(line_status&BREAK_ERR) strcat(report," Error por recibir trama de ceros\n");
if(line_status&FRAM_ERR) strcat(report," Bit de paro erroneo en Rx\n");
if(line_status&OVER_ERR) strcat(report," Dato en Rx sobrescrito\n");
if(line_status&PAR_ERR) strcat(report," Error de paridad en Rx\n");
if(line_status&RX_DATA_RDY){
  char data_over=' ';
  deshabilita_recepcion_asincrona();
  data_over = rtio_inb(RX_REG_COM2);
  habilita_recepcion_asincrona();
  strcat(report," Dato en buffer Rx ");
  strcat(report,data_over);
  strcat(report,"\n");
 }
else if(line_status&TX_RDY) strcat(report," Posibilidad para transmitir\n");
}

void estado_puerto(w,client_data,call_data)
Widget  w;
caddr_t client_data;
caddr_t call_data;
{
static    char report[200];
/* Lee los registros */
lee_registros(report);
/* Crea ventana de estado */
ventana_estado_puerto(w,report);
}
```

```
void serial_port_CB(w,client_data,call_data)
Widget  w;
int     client_data;
caddr_t call_data;
{
static Widget  BullB;
Position x_p,y_p;
Arg     args[2];
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(XtParent(w),args,2);
XtSetSensitive(w,FALSE);
BullB = CreateBulletinBoard("PUERTO SERIE",w,Estructura_modulo_puerto_serie,
        XtNumber(Estructura_modulo_puerto_serie),x_p,y_p);
XtManageChild(BullB);
}

int transmite(size,str)
int  size;
char *str;
{
int conn=0,line_status=0,error;
ushort  filter=0xFF7F; /* filter = 1111 1111 0111 1111 */
char datos[10];
deshabilita_recepcion_asincrona();
ioctl(fildes,TCGETA,&CONFIGURACION);
CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag & filter );
if(ioctl(fildes,TCSETA,&CONFIGURACION)<0){
   habilita_recepcion_asincrona();
   return PAR_FILDES_ERR;
  }
conn=conector();
if(conn==CONN_ERR){
   CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | CREAD );
   ioctl(fildes,TCSETA,&CONFIGURACION);
   habilita_recepcion_asincrona();
   return CONN_ERR;
```

```
 }
line_status=inb(L_S_REG_COM2);
if( line_status & RX_DATA_RDY ){
 error=DATA_ERR; /* Error dato en el puerto */
 if( line_status & OVER_ERR ){
   error=OVER_ERR; /* Dato en el puerto sobrescrito */
   if((line_status&PAR_ERR)||(line_status&FRAM_ERR)||(line_status&BREAK_ERR))
     error=PAR_ERR; /* Error de dato no valido */
 }
 CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | CREAD );
 ioctl(fildes,TCSETA,&CONFIGURACION);
 habilita_recepcion_asincrona();
 return error;
 }
 while(size!=0){
   int bit_tx=0;
   outb(TX_REG_COM2,*str);
   size--;
   str++;
   while(bit_tx == 0 && size > 0 ){
      bit_tx = ( inb(L_S_REG_COM2) & TX_RDY );
    }
 }
CONFIGURACION.c_cflag = ( CONFIGURACION.c_cflag | CREAD );
ioctl(fildes,TCSETA,&CONFIGURACION);
habilita_recepcion_asincrona();
return 0;
}

void TRANSM(w,ID_TX,call_data)
Widget  w;
int     ID_TX;
caddr_t call_data;
{
int     tx_error=0;
char    *str;
static Widget  InfoDialog;
```

```
Arg    args[4];
switch (ID_TX){
  case TX_BYTE: /* Tx Byte */
       if(DISPLAY = =ON)
            str = XmTextGetString(XmSelectionBoxGetChild(w,
                                        XmDIALOG_TEXT));
       else if(DISPLAY = =OFF) strncpy(str,buff_tx,1);
       tx_error = transmite(1,str);
       break;
  case TX_STRING: /* Tx String */
       if(DISPLAY = =ON)
            str = XmTextGetString(XmSelectionBoxGetChild(w,
                                        XmDIALOG_TEXT));
       else if(DISPLAY = =OFF) strncpy(str,buff_tx,strlen(buff_tx));
       tx_error = transmite(strlen(str),str);
       break;
  case TX_FILE: /* Tx File */
       if(DISPLAY = =ON)
            str = XmTextGetString(XmSelectionBoxGetChild(w,
                                        XmDIALOG_TEXT));
       else if(DISPLAY = =OFF) strncpy(str,buff_tx,strlen(buff_tx));
       break;
  }
if((tx_error! =0)&&(DISPLAY = =ON)){
  XtSetArg(args[0],XmNnoResize,TRUE);
  XtSetArg(args[1],XmNautoUnmanage,FALSE);
  switch(tx_error){
       case -1:
            XtSetArg(args[2],XmNmessageString,
                 XmStringCreateLtoR(" No acceso al archivo
                 descriptor ",XmSTRING_DEFAULT_CHARSET));
            break;
       case CONN_ERR:
            XtSetArg(args[2],XmNmessageString,
                 XmStringCreateLtoR(" No existe conector
                  ",XmSTRING_DEFAULT_CHARSET));
            break;
```

```
    case DATA_ERR:
        XtSetArg(args[2],XmNmessageString,
                XmStringCreateLtoR(" Habia dato en el puerto
                ",XmSTRING_DEFAULT_CHARSET));
        break;
    case OVER_ERR:
        XtSetArg(args[2],XmNmessageString,
                XmStringCreateLtoR(" Dato en el puerto sobrescrito
                ",XmSTRING_DEFAULT_CHARSET));
        break;
    case PAR_ERR:
        XtSetArg(args[2],XmNmessageString,
                XmStringCreateLtoR(" Dato en el puerto\n
                perdido por errores ",XmSTRING_DEFAULT_CHARSET));
        break;
 }
 XtSetArg(args[3],XtNtitle," Error ");
 InfoDialog = XmCreateInformationDialog(XtParent(w),
          NULL,args,XtNumber(args));
 XtUnmanageChild(XmMessageBoxGetChild(InfoDialog,
          XmDIALOG_HELP_BUTTON));
 XtUnmanageChild(XmMessageBoxGetChild(InfoDialog,
          XmDIALOG_CANCEL_BUTTON));
 XtAddCallback(InfoDialog,XmNokCallback,Destroy_CB,w);
 XtManageChild(InfoDialog);
}
}

void win_tx_CB(w,ID_TX,call_data)
Widget  w;
int     ID_TX;
caddr_t call_data;
{
static Widget    Dialog_prompt;
Arg      args[7];
Position  x_p=0,y_p=0;
XtSetSensitive(w,FALSE);
```

```
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(XtParent(w),args,2);
XtSetArg(args[0],XmNnoResize,TRUE);
XtSetArg(args[1],XmNautoUnmanage,FALSE);
XtSetArg(args[2],XmNokLabelString,XmStringCreateLtoR(" Continua ",
      XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[3],XmNcancelLabelString,XmStringCreateLtoR(" Salir ",
      XmSTRING_DEFAULT_CHARSET));
switch(ID_TX){
      case TX_BYTE:
            XtSetArg(args[4],XmNselectionLabelString,
                  XmStringCreateLtoR(" Byte: ",
                  XmSTRING_DEFAULT_CHARSET));
            XtSetArg(args[5],XmNdefaultPosition,FALSE);
            XtSetArg(args[6],XtNtitle," BYTE ");
            break;
      case TX_STRING:
            XtSetArg(args[4],XmNselectionLabelString,
                  XmStringCreateLtoR(" Cadena: ",
                  XmSTRING_DEFAULT_CHARSET));
            XtSetArg(args[5],XmNdefaultPosition,FALSE);
            XtSetArg(args[6],XtNtitle," CADENA ");
            break;
      case TX_FILE:
            XtSetArg(args[4],XmNselectionLabelString,
                  XmStringCreateLtoR(" Archivo: ",
                  XmSTRING_DEFAULT_CHARSET));
            XtSetArg(args[5],XmNdefaultPosition,FALSE);
            XtSetArg(args[6],XtNtitle," ARCHIVO ");
           break;
      }
Dialog_prompt = XmCreatePromptDialog(XtParent(w),NULL,args,XtNumber(args));
if(ID_TX==TX_BYTE){
      XmTextSetMaxLength(XmSelectionBoxGetChild(Dialog_prompt,
                  XmDIALOG_TEXT),1);
```

```
        XtUnmanageChild(XmSelectionBoxGetChild(Dialog_prompt,
                        XmDIALOG_HELP_BUTTON));
        XtAddCallback(Dialog_prompt,XmNcancelCallback,Destroy_CB,w);
        XtAddCallback(Dialog_prompt,XmNokCallback,TRANSM,TX_BYTE);
        XtMoveWidget(Dialog_prompt,x_p+10,y_p+33);
      }
else {
        XtUnmanageChild(XmSelectionBoxGetChild(Dialog_prompt,
                        XmDIALOG_HELP_BUTTON));
XtAddCallback(Dialog_prompt,XmNcancelCallback,Destroy_CB,w);
if(ID_TX==TX_STRING)
        XtAddCallback(Dialog_prompt,XmNokCallback,TRANSM,TX_STRING);
else if(ID_TX==TX_FILE)
        XtAddCallback(Dialog_prompt,XmNokCallback,TRANSM,TX_FILE);
        XtMoveWidget(Dialog_prompt,x_p+82,y_p+33);
   }
XtManageChild(Dialog_prompt);
}

void tx_CB(w,client_data,call_data)
Widget  w;
int     client_data;
caddr_t call_data;
{
static Widget   BullB;
Position x_p,y_p;
Arg     args[2];
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(XtParent(w),args,2);
XtSetSensitive(w,FALSE);
BullB = CreateBulletinBoard(" TRANSMITE ",w,Estructura_modulo_transmite,
        XtNumber(Estructura_modulo_transmite),x_p,y_p);
XtManageChild(BullB);
}
```

```
void modulo_de_comunicaciones(w,client_data,call_data)
Widget  w;
caddr_t  client_data;
caddr_t  call_data;
{
static Widget  BullB;
Position x_p,y_p;
Arg     args[2];
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(w,args,2);
XtSetSensitive(w,FALSE);
BullB = CreateBulletinBoard(" COMUNICACIONES
",w,Estructura_modulo_Comunicaciones,XtNumber(Estructura_modulo_Comunic
aciones),x_p,y_p);
XtManageChild(BullB);
}

void conf_loopback(w,ok,call_data)
Widget  w;
Widget  ok;
caddr_t call_data;
{
configura_puerto(w,YES,call_data);
XtUnmanageChild(XmMessageBoxGetChild(wid_loopback,XmDIALOG_CANCEL_
BUTTON));
}

void Destroy_Widget_loopback(w,ok,call_data)
Widget  w;
Widget  ok;
caddr_t call_data;
{
XtDestroyWidget(w);
habilita_recepcion_asincrona();
}
```

```
void ventana_autoprueba_parametros_iniciales(result)
char *result;
{
Widget ok_widget;
Arg    args[9];
XtSetArg(args[0],XmNnoResize,TRUE);
XtSetArg(args[1],XmNautoUnmanage,FALSE);
XtSetArg(args[2],XmNokLabelString,XmStringCreateLtoR(" CONFIGURAR ",
        XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[3],XmNcancelLabelString,XmStringCreateLtoR(" Continuar ",
        XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[4],XmNmessageString,XmStringCreateLtoR(result,
        XmSTRING_DEFAULT_CHARSET));
XtSetArg(args[5],XmNdefaultPosition,FALSE);
XtSetArg(args[6],XtNtitle," AUTOPRUEBA ");
XtSetArg(args[7],XmNx,130);
XtSetArg(args[8],XmNy,50);
w i d _ i o o p b a c k =
XmCreateInformationDialog(Port_parent,NULL,args,XtNumber(args));
XtAddCallback(wid_loopback,XmNcancelCallback,Destroy_Widget_loopback,NULL);
ok_widget = XmMessageBoxGetChild(wid_loopback,XmDIALOG_OK_BUTTON);
if( ENBL_COM == ON ){
      if((fildes=open(COM2,O_RDWR|O_NDELAY))<0)    ENBL_COM    =
OPEN_FILDES_ERR;
      else {
        CONFIGURACION.c_cflag = 0;
        CONFIGURACION.c_cflag = ( CREAD | CS8 | B2400 );
         if(ioctl(fildes,TCSETA,&CONFIGURACION)<0)    ENBL_COM    =
PAR_FILDES_ERR;
      }
      if(ENBL_COM == ON)
        XtAddCallback(wid_loopback,XmNokCallback,conf_loopback,ok_widget);
      else
        XtUnmanageChild(ok_widget);
  }
else
```

```
XtUnmanageChild(ok_widget);
XtUnmanageChild(XmMessageBoxGetChild(wid_loopback,
                    XmDIALOG_HELP_BUTTON));
}

int reporte_autoprueba_parametros_iniciales(rx,tx,result)
char *rx, *tx, *result;
{
sprintf(result,"\
      Los valores de la Autoprueba son :\n\
\n\
* Velocidad en Bauds ..... 9600\n\
* Bits de dato ..... 8\n\
* Bits de paridad ..... PAR\n\
* Bits de paro ..... 1\n\n");
if(ENBL_COM == ON){
    strcat(result," Dato transmitido: ");
    strcat(result,tx);
    strcat(result,"\n");
    strcat(result," Dato recibido: ");
    strcat(result,rx);
    strcat(result,"\n");
    strcat(result," Edo. del Puerto: ");
    if(strcmp(tx,rx) == =0) strcat(result," Bueno ");
    else{
        strcat(result," Malo ");
        ENBL_COM = RX_LOOP_ERR;
        return 0;
      }
    strcat(result,"\n\n\
      Los parametros iniciales son :\n\
\n\
* Velocidad en Bauds ..... 2400\n\
* Bits de dato ..... 8\n\
* Bits de paridad ..... Ninguna\n\
* Bits de paro ..... 1\n\n\
      Si desea cambiar los parametros \n\
```

```
      presione el boton de CONFIGURA\n");
  }
else if(ENBL_COM==ENBL_IO_ERR){
   strcat(result," Error al habilitar el bus IO\n");
   return 0;
 }
}

int realiza_autoprueba(tx,rx)
char *tx,*rx;
{
int  count1=0;
struct
     {
       unchar      d_l_l; /* Divisor Latch Least */
       unchar  d_l_m;     /* Divisor Latch Most */
       unchar      l_c_r; /* Line Control Register */
       unchar      m_c_r;       /* Modem Control Register */
     } loopback;
rx[0]='\0';
loopback.d_l_l = (unchar) 0x0C;       /* Valor bajo para 9600 Bauds */
loopback.d_l_m = (unchar) 0x00;       /* Valor alto para 9600 Bauds */
loopback.l_c_r = (unchar) 0x80; /* Valor para accesar d_l_l y d_l_m */
loopback.m_c_r = (unchar) 0x1F;       /* Valor para loopback */
if(rtioset(RT_IO_ENABLE)<0){
       ENBL_COM = ENBL_IO_ERR;
       return 0;
    }
       outb(L_C_REG_COM2,loopback.l_c_r);
       outb(D_L_L_COM2,loopback.d_l_l);
       outb(D_L_M_COM2,loopback.d_l_m);
       outb(M_C_REG_COM2,loopback.m_c_r);
       loopback.l_c_r = (unchar) 0x1F; /* Parametros restantes del puerto */
       outb(L_C_REG_COM2,loopback.l_c_r);
       while(strlen(tx)!=0){
             int bit_rx=0,count2=0;
             outb(TX_REG_COM2,*tx);
```

```
            do{
                  bit_rx = ( inb(L_S_REG_COM2) & 0x01 );
                  count2++;
            }while((bit_rx==0) || (count2<=100000));
            count2 = 0;
            rx[count1] = (char) inb(RX_REG_COM2);
            tx++;
            count1++;
            rx[count1]='\0';
      }
      tx -= count1;
      loopback.m_c_r = (unchar) 0x0F; /* Deshabilita loopback */
      outb(M_C_REG_COM2,loopback.m_c_r);
}

void activa_ventana_autoprueba()
{
alarm(0);
XtManageChild(wid_loopback);
if(ENBL_COM == ON){
  XtSetSensitive(Port_button,TRUE);
  memset(buff_rx_int,0x00,sizeof(buff_rx_int));
 }
}

void estado_configura(w,client_data,call_data)
Widget w;
caddr_t client_data;
caddr_t call_data;
{
static Widget   BullB;
Position x_p,y_p;
Arg     args[2];
XtSetArg(args[0],XmNx,&x_p);
XtSetArg(args[1],XmNy,&y_p);
XtGetValues(w,args,2);
XtSetSensitive(w,FALSE);
```

```
BullB = CreateBulletinBoard(" COMUNICACIONES",w,Estructura_configura_estado,
                    XtNumber(Estructura_configura_estado),x_p,y_p);
XtManageChild(BullB);
}

void puerto_serie()
{
Port_button = XmCreateCascadeButton(Port_parent," Puerto ",NULL,0);
      XtManageChild(Port_button);
      XtSetSensitive(Port_button,FALSE);
if (DISPLAY)
  XtAddCallback(Port_button,XmNactivateCallback,modulo_de_comunicaciones,0);
else
 XtAddCallback(Port_button,XmNactivateCallback,estado_configura,0);
}

autoprueba_parametros_iniciales(parent)
Widget parent;
{
Widget button;
char rx[20],result[1500],*tx = "Test Loopback";
Port_parent = parent;
realiza_autoprueba(tx,rx);
reporte_autoprueba_parametros_iniciales(tx,rx,result);
ventana_autoprueba_parametros_iniciales(result);
alarm(7);
sigset(SIGALRM,activa_ventana_autoprueba);
}
```

k).- Venix/386, System V Release 3.2.4: Installation and Release Notes. VenturnCom Inc. Cambridge, 1992.

l).- Valley, John J. Unix Programmer's Guide Reference. QUE corporation. United States of America, 1991.

m).- Barkakati, Nabajyoti. X Window System Programming. Sams. United States of America, 1991.

n).- Godfrey, Terry. Lenguaje ensamblador para microcomputadoras IBM para principiantes y avanzados. Prentice-Hall Hispanoamerica. México, 1991.

o).- AT&T. Unix System V/386 Release 4: Programmer's Reference Manual. Unix Press, a Prentice-Hall title. Englewood Cliffs, N.J. 1990.

p).- AT&T. Unix System V/386 Release 4: Programmer's Guide: STREAMS. Unix Press, a Prentice-Hall title. E. C., N.J. 1990.

q).- VenturCom. Venix: Real Time Programmer's Guide. Unix Press. N.J. 1991.

a).- Mc Gover, Tom. Data Communications Concepts and Applications. Prentice-Hall. Canada Inc, 1989.

b).- Schwartz, Mitcha. Transmisión de información, modulación y ruido. McGaw-Hill. México, 1987.

c).- Black, Uless. Redes de computadoras: protocolos, redes e interfaces. Microbit. México, 1990.

d).- NS16450/INS8250A/NS16C450/INS82C50A Asynchronous communications elements. National Semiconductor Corp. Sta. Clara, 1985.

e).- EIA Standard RS-232-C. INTERFACE BETWEEN DATA TERMINAL EQUIPMENT AND DATA COMMUNICATION EQUIPMENT EMPLOYING SERIAL BINARY DATA INTERCHANGE. Electronic Industries Association, Engineering Department. Washington, 1969.

f).- Jourdain, Robert. Programmer's problem solver for IBM PC, XT & AT. Brady Communications Company. New York, 1986.

g).- CS-386/40. User's Guide. Hyundai Electronics Industries Co. Korea, 1992.

h).- Sommerville, I. Ingeniería de software. Addison W. Iberoamericana, 1988.

i).- Donovan, John J. Programación de sistemas. El Ateneo. México, 1979.

j).- Dahl, O. J; Dijkstra, E. W. Structured programming. Academic Press Inc. New York, 1978.

INDICE

PAG.