CINVESTAV-IPN
Biblioteca de Ingeniería Eléctrica
FB0000029636

CENTRO DE INVESTIGACION Y DE
ESTUDIOS AVANZADOS DEL
I. P. N.
BIBLIOTECA
INGENIERIA ELECTRICA

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS

DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

"SISTEMA SUPERVISOR DE PROCESOS EN UNA COMPUTADORA PERSONAL"

Tesis que presenta el Ing. Felipe Verdalet Guzmán para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERIA ELECTRICA. Trabajo dirigido por el Dr. Renato Barrera Rivera.

CENTRO DE INVESTIGACION Y DE
ESTUDIOS AVANZADOS DEL
I. P. N.
BIBLIOTECA
INGENIERIA ELECTRICA

Becario del COSNET

Mexico, D.F., Septiembre de 1987.

XM

CLASIF. 87.19
ADQUIS. BL 10813
FECHA: 28-IV-81
PROCED. Don
$

Deseo expresar mi agradecimiento a las siguientes personas e instituciones:

Consejo del Sistema Nacional de Educación Tecnológica.

Centro de Investigación y de Estudios Avanzados Del Instituto Politécnico Nacional.

Dr Renato Barrera Rivera, por la acertada dirección de esta tesis.

Dr. Manuel Guzmán Renteria y Dr. Armando Maldonado Talamantes, pues sus comentarios y sugerencias fuerón muy oportunos y constructivos.

Oscar Nava Garcia y Ana Emilia Verdalet de Nava, por la gran ayuda que me han prestado.

CENTRO DE INVESTIGACION Y DE
ESTUDIOS AVANZADOS DEL
I. P. N.
BIBLIOTECA
INGENIERIA ELECTRICA

Dedico el presente trabajo a:

mis padres:

Eliezer Verdalet López y Emiliana Guzmán de Verdalet, gracias a sus consejos y apoyo he podido siempre salir adelante.

mis hermanos:

Elie
Eliezer
Ana Emilia
Iñigo
Manolo
Ma. Saadia
Cuauhtemoc
Eugenio

La meta que cada uno de nosotros logra es un triunfo de todos.

mi esposa:

Raquel

en verdad que ha sido dificil el tener por compañero a un estudiante, y aun asi apoyarme siempre.

mis hijos:

Felipe y Valeria Raquel

han sido mi mayor estímulo para salir adelante.

CENTRO DE INVESTIGACION Y DE
ESTUDIOS AVANZADOS DEL
I. P. N.
BIBLIOTECA
INGENIERIA ELECTRICA

CONTENIDO

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL I. P. N.
BIBLIOTECA
INGENIERIA ELECTRICA

CAPITULO 1

INTRODUCCION.

1.1 .- Definición de Supervisor de Procesos

Un sistema supervisor se encarga de llevar un seguimiento a ciertas variables generadas en algún proceso.

Como resultado de este seguimiento es posible llevar a cabo diferentes acciones, por ejemplo:

- Llevar un historial del comportamiento de cada variable.

- Alertar al operador del sistema por medio de una alarma cuando una variable se salga del rango especificado para esta.

- Enviar los valores de las variables a un modulo de control, el cual se encarga de realizar los ajustes necesarios para el buen funcionamiento del proceso supervizado.

Como proceso se entiende a un conjunto de acciones interrelacionadas y organizadas para originar cambios en un sistema.

Un ejemplo de un proceso se muestra en la fig. 1.1: en ella se representa un sistema de generación de vapor compuesto por la fuente de calor, la tuberia que transporta el agua y la tuberia que transporta el vapor; como resultado de la aplicación de el calor sobre la tuberia que trasporta el agua se genera vapor.

En este proceso las variables que nos interesan supervisar son principalmente la presión en la tuberia y la temperatura. La lectura de estas variables se realiza a través de Sensores(instrumentos que permiten medir la temperatura, presión, flujo, etc.) e Interruptores(instrumentos que solo reportan dos estados, encendido ó apagado)

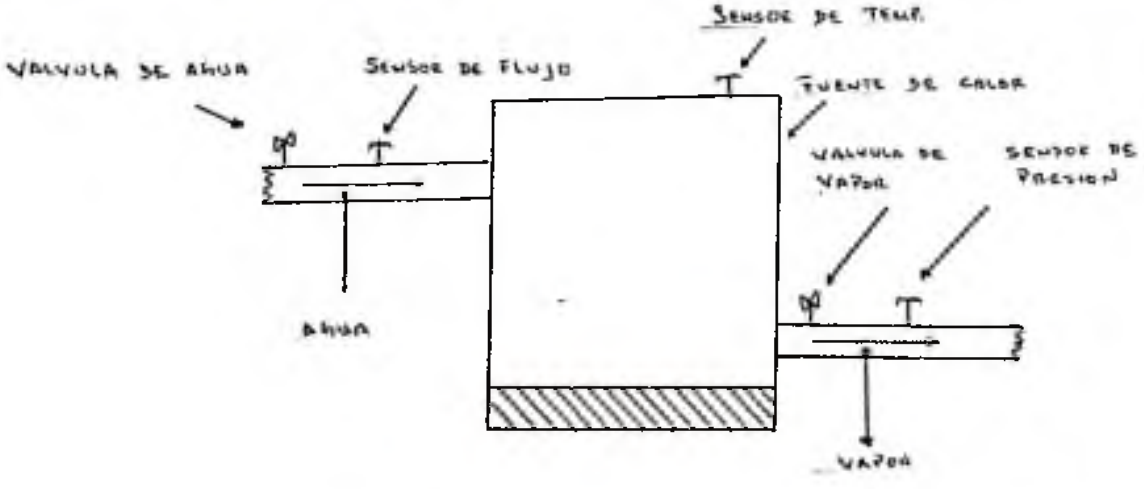


Fig. 1.1 PROCESO DE GENERACION DE VAPOR

En la fig. 1.2 se muestra un diagrama a bloques de un sistema supervisor. En éste diagrama el ambiente de el sistema supervisor esta formado por el sistema bajo supervisión y el operador de el sistema. El ambiente(como ambiente entendemos todo aquello que afecta ó es afectado por el sistema) interactúa con el sistema de control a través de la frontera , que se compone de las alarmas, la consola, la impresora y los sensores [Gerez 84].

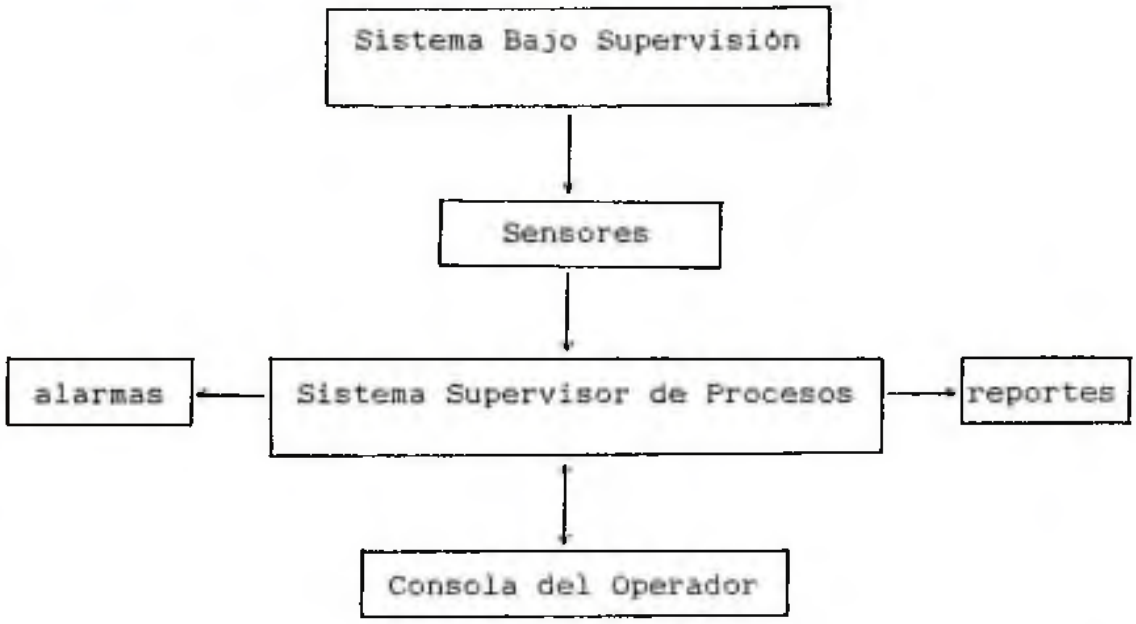


Fig. 1.2 DIAGRAMA DE BLOQUES DE UN SISTEMA SUPERVISOR

1.2.- Objetivo:

Se trata de diseñar y programar un Sistema Supervisor de Procesos con las siguientes características:

- Se instrumentará en una microcomputadora P.C. tipo XT, ST ó AT.
- Debe tener capacidad para supervisar variables de tipo digital.
- Debe ser posible añadir al sistema módulos para supervisar variables de tipo analogico, asi como módulos de Control.
- Los resultados de la supervisión se mostrarán al operador por pantalla.
- Se intrumentarán un conjunto de procedimientos para el manejo de procesos concurrentes mediante el uso de Señales. A este conjunto de procedimientos le llamaremos "Kernel" ó "Nucleo de Concurrencia".
- Se utilizara el lenguaje Modula-2 para la programación del

sistema.

1.3 EL Supervisor y su Medio Ambiente de Prueba

En la fig. 1.3 se muestra el esquema que se utilizara para probar el sistema diseñado. Se ha elegido la red de recolección y transmisión de datos construida en el Depto. de Ingenieria Electrica/ Sección Comunicaciones con las siguientes caracteristicas:

- Cuatro ramas multi-punto cada una con capacidad de hasta 128 Unidades de Terminales Remotas(UTR's).

- Un Procesador de Comunicaciones (PC) que maneja estas cuatro ramas multi-punto.

- El Procesador de Comunicaciones tiene un puerto serie para comunicarse por linea directa con el computador donde se encuentra residente el Sistema Supervisor.

- La comunicación entre UTR's y PC es via cable telefonico y con un modem(instrumento que se utiliza para establecer la comunicación entre dispositivos a través de lineas analogicas) por cada UTR y PC.

- La velocidad de transmisión entre PC y Supervisor es de 9600 bps.

- La velocidad de transmisión entre UTR's y PC es de 1200 bps.

- El protocolo para transferencia de información entre UTR's y PC y entre PC y Supervisor se encuentra programado en EPROM.

La configuración usada con el Sistema Supervisor de Procesos fue la siguiente:

- Dos UTR's

- Un Procesador de Comunicaciones.

Cada UTR tiene las siguientes funciones:

- Establecer la comunicación a través de la linea multi-punto con el PC.

- Adquirir los datos tanto analógicos como digitales de los puntos de monitoreo que tenga asignados e introducirlos en la base de datos que se encuentra en la memoria primaria. Esta base de datos se mantendra actualizada con los valores mas recientes.

- Controlar los puntos(sensores ó interruptores) que se le

asignen. La información necesaria para realizar esto, se encuentra en la base de datos en la memoria primaria. Esta base de datos se accesa cuando se detecta un mensaje control para la UTR en cuestión.

El Procesador de Comunicaciones tiene las siguientes funciones asignadas:

- Controlar todos los canales asignados al procesador en cuanto al poleo.

- Comunicarse con el Sistema Supervisor por la línea directa.

- Con base en los procesos de comunicación con las remotas deberá mantener una base de datos en memoria primaria, misma que sera accesada cuando asi se requiera por el Supervisor de Procesos.

El esquema general del Sistema Supervisor es del tipo maestro-esclavo es decir el Procesador de Comunicaciones se limita a responder las preguntas realizadas por el Supervisor de Procesos.

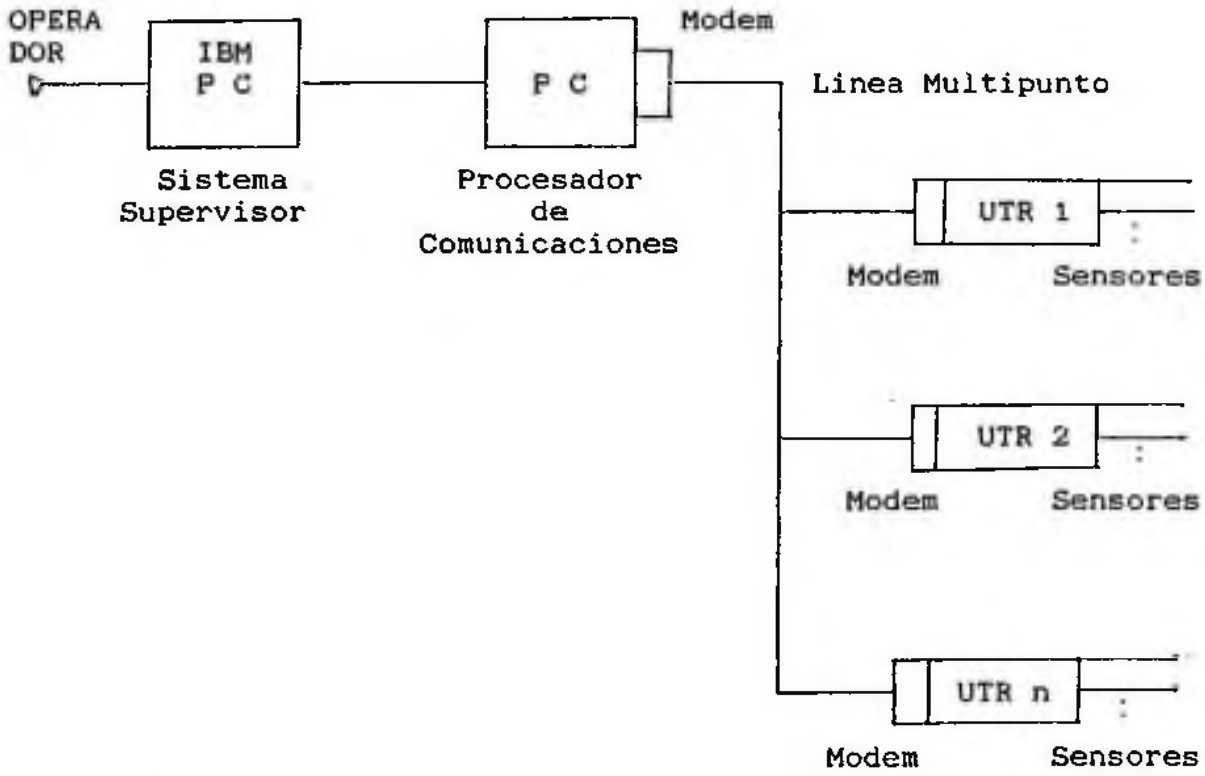


Fig. 1.3 SISTEMA SUPERVISOR Y SU MEDIO AMBIENTE

1.4 Descripción del Sistema Desarrollado

El trabajo consistió en los siguientes puntos:

- Desarrollar un sistema capaz de efectuar una supervisión sobre variables ya sea analógicas o digitales, las cuales seran recibidas a través de el puerto serie RS-232 primario de la P.C.

- Mostrar los resultados de la evaluación de la supervisión por pantalla.

- Utilizar el concepto de mensajes para la interacción Operador - Supervisor - Proceso. Esto es que el operador puede indicar al supervisor lo que debe ser monitoreado mediante mensajes.

El patrón de mensajes es el siguiente:

- El supervisor envia un mensaje al procesador de comunicaciones para pedir la información solicitada por el operador del sistema.

- El procesador de comunicaciones contesta a la pregunta del supervisor.

- El Supervisor recibe la respuesta, la analiza y muestra el resultado del análisis al operador por medio del video de la consola.

- En caso de ser necesario se activa un mensaje de alarma, el cual se muestra al operador a través del video.

Dado que un supervisor necesita tener caracteristicas de tiempo real, es necesario instrumentar el sistema con base en procesos concurrentes y control directo de dispositivos claves. Para lograr esto se utiliza el sistema de interrupciones proporcionado por la microcomputadora.

En capitulos posteriores hablaremos de como se manejó cada uno de estos puntos.

CAPITULO 2

NUCLEO DE CONCURRENCIA

En este capitulo se introducen conceptos de multiprogramación, asi como la definición de las primitivas usadas por el nùcleo de concurrencia utilizado en el sistema. Algunos conceptos de Modula-2 se encuentran en el apendice A.

2.1 Procesos

Un programa secuencial especifica la ejecución secuencial de una lista de instrucciones; a esta ejecución se le llama proceso. Un programa concurrente especifica uno o mas subprogramas secuenciales que pueden ser ejecutados simultaneamente como procesos paralelos.

En lo que resta del trabajo, el significado del término proceso se refiere a la definición que se acaba de establecer, la cual corresponde a un proceso en el contexto de programación.

Un programa concurrente puede ser ejecutado ya sea permitiendo a los procesos compartir uno (o varios) procesadores ó corriendo cada proceso en su propio procesador. La primera forma es conocida como multiprogramación; ésta es soportada por un "kernel" del sistema operativo[Dijkstra 68] que alterna los procesos en el procesador o procesadores. La segunda forma es conocida como multiprocesamiento si los procesadores comparten una memoria comun, ó como procesamiento distribuido si los procesadores estan conectados por medio de una red[Andrews 83].

2.2 Comunicación y Sincronización

Para que procesos concurrentes se ayuden entre si debe existir comunicación y sincronización entre ellos.

La comunicación permite que la ejecución de un proceso influya en la forma que otro se ejecute. La comunicación entre procesos se basa ya sea en el uso de variables compartidas(variables que pueden ser referenciadas por mas de un proceso) ó en el pase de mensajes.

Cuando se comunican procesos es necesaria la sincronización entre los mismos, debido a que para comunicar un proceso éste debe ejecutar alguna acción que los otros detecten. Esta acción puede ser cambiar el valor de una variable ó enviar un mensaje.

Esto sólo funciona si los eventos 'ejecuta acción' y 'detecta acción' suceden en ese orden. Asi pues la sincronización puede verse como un conjunto de restricciones en el ordenamiento de los eventos.

Los mecanismos de sicronización controlan la interferencia en dos caminos:

- Primero: Se puede retardar la ejecución de un proceso hasta que una condición dada sea cierta, y de esta manera se asegura que la precondición de la subsequente instrucción sea cierta.

- Segundo: Se puede usar un mecanismo de sincronización para asegurar que un conjunto de instrucciones se ejecute de forma indivisible "atomica"(i.e. una vez que se haya comenzado a ejecutar las instrucciones, no se interrupa hasta que termine su ejecución).

La atomicidad de un bloque de instrucciones elimina la posibilidad de que instrucciones de otros procesos interfieran con sus aserciones.

Cuando deseamos expresar una computación concurrente hay tres preguntas basicas que se deben contestar:

- Como especificamos la ejecución concurrente ?
- Que forma de comunicacióusamos entre los procesos?
- Que mecanismo de sincronización usamos?

Los mecanismos de sincronización pueden ser tales como el ordenamiento de eventos ó el control de interferencias[Andrews 83]

2.3 Corrutinas

Las corrutinas son utilizadas para lograr la concurrencia en este sistema. Debido a que éstas se contemplan en la definición del lenguaje Modula-2, las veremos con detalle.

Definimos corrutina como una serie de instrucciones, las cuales se ejecutan en forma secuencial y que tienen asignadas un area de memoria en donde se guarda el contexto de las mismas. Se entendiende como contexto de una corrutina a la información necesaria para que en un momento dado se continue la ejecución de las instrucciones de ella como si jamas hubiera sido interrumpida.

En contraste con la relación asimétrica de una rutina principal y una subrutina, existe simetría completa entre varias corrutinas que se llaman mutuamente[Knuth 73].

Mientras que una subrutina se ejecuta desde el principio cada vez que se invoca, una corrutina se ejecuta a partir de la instrucción en que se encontraba al ser abandonada, y no necesariamente regresa a la corrutina que la invocó. Por ejemplo:

```
tiempo        Corrutina A        Corrutina B        Corrutina C

              instrucción 1
              instrucción 2
              Transfer(A,B)          -|
                                  instrucción 1
                                  instrucción 2
                                  Transfer(B,A);
              instrucción 4
              instrucción 5
              Transfer(A,C)
                                                  instrucción 1
                                                  instrucción 2
                                                        "
```

En este caso suponiendo que se inicia ejecutando la corrutina A, al momento que se ejecuta la instrucción Transfer(A,B) se realiza un cambio de contexto. Es decir se guardan los valores de las variables locales, el contenido de los registros, el apuntador de pila y el apuntador de instrucciones. Se pasa el control del procesador a la corrutina B. Cuando se esta en B, y se ejecuta la instrucción Transfer(B,A) se realiza lo mismo, se asigna el procesador a la corrutina A. Una corrutina conserva el procesador a menos que se ejecute una instrucción Transfer.

2.4 instrumentación del Núcleo de Concurrencia.

El núcleo de concurrencia usado en este sistema es el definido en el módulo procesos recomendado por Wirth [Wirth, 1982]. Este núcleo tiene la ventaja de ofrecer las funciones necesarias para la multiprogramación en un alto nivel de abstracción. Practicamente sin necesidad de construcciones adicionales del lenguaje.

En el núcleo se definen los siguientes procedimientos: StartProcess, SEND, WAIT, Awaited, Init.

```
StartProcess(P:PROC;n:CARDINAL);
(* comienza un proceso concurrente con el programa P y un
espacio de trabajo de tamaño n, PROC es un tipo estandard
definido como PROC = PROCEDURE *)

SEND(VAR s : SIGNAL);
(* un proceso que esta esperando por s es resumido *)

WAIT( VAR s:SIGNAL);
(* espera hasta que algun otro proceso envie s *)

Awaited(s:SIGNAL):BOOLEAN
(* Awaited(s) = " al menos un proceso esta esperando por  s"
*)
```

```
Init(VAR s:SIGNAL);
(* inicializa una variable tipo signal *)
```

Veamos cada uno de estos procedimientos en detalle.

Un llamado a StartProcess(P,n) comienza la ejecución de un proceso, expresado por un procedimiento P. Cada proceso necesita un cierto espacio de trabajo en la memoria para guardar sus variables de trabajo. El espacio de trabajo en terminos de palabras esta dado por un entero "n"; este espacio se escoge dependiendo del numero de variables locales y llamadas locales usadas en este proceso.

Un valor aproximado de "n" se calcula a partir de la suma de los tamaños de los parametros de los procedimientos con los requerimientos de memoria de los objetos declarados como locales, y con los tamaños del area de trabajo necesarios para cualquier procedimiento declarado internamente. Una aproximación de los tamaños de memoria para las diferentes variables y constantes es usar cuatro palabras para cada constante, variable, elemento de arreglo, campo y apuntador en el procedimiento. Tambien existe un requerimiento de memoria extra el cual se define por la versión del lenguaje usado. Por ejemplo en la versión de Modula-2 de Logitech se necesitan 400 bytes de "overhead".

Al momento de inicializar el módulo procesos, el programa principal se considera como un proceso mas de tal modo que cuando llamamos al procedimiento "StartProcess" realizamos un cambio de contexto del proceso actual al proceso que se acaba de definir y se comienza a ejecutar este.

La comunicación entre procesos se puede realizar en dos formas distintas: Por medio de variables compartidas ó a través de señales.

Las variables comunes son usadas para transferir datos entre procesos, y resolver el problema de lograr una cooperación armoniosa. Ningun proceso deberá tener acceso a variables comunes mientras otro proceso esté realizando alguna acción critica sobre estas. Una solución razonable es la de encapsular las variables comunes en un módulo, lo cual garantize la exclusión mutua entre procesos. Tal módulo que mas tarde sera discutido es llamado "Monitor".

Las señales exportadas como tipos de datos de el Módulo Procesos no llevan datos por si mismas, pero sirven para sincronizar procesos.

Solo dos operaciones son aplicables a las señales aparte de la inicialización de estas: el envio de una señal de un proceso a otro, ó la espera por un proceso de una señal enviada por un segundo.

Todas las señales denotan una cierta condición o estado entre las variables de los programas, y enviar una señal deberá

implicar que esta condición ha sido cumplida. Un proceso que recibe por una señal esperada puede entonces asumir que esta condición ha sido cumplida y continuar su trabajo.

El envio de una señal reactiva a lo sumo un proceso. El envio de una señal que nadie espera se considera una acción nula.

En este sistema solo se realiza un cambio de contexto cuando se efectua una operación WAIT o SEND.

Un monitor es un módulo el cual garantiza la exclusión mutua entre procesos y por tanto puede garantizar la integridad de sus datos locales.

A continuación se verán los procesos TRANSFER y NEWPROCESS:

PROCEDURE TRANSFER(VAR fuente,destino:PROCESS);

El llamado a este procedimiento causa que el proceso que se esta ejecutando sea suspendido (en el orden adecuado para ser reanudado mas tarde en la siguiente instrucción después del transfer) y el proceso se reanuda en el punto en que estaba suspendido.

Para crear una corrutina se llama a el procedimiento NEWPROCESS.

PROCEDURE NEWPROCESS(P:PROC;A:ADDRESS;n:CARDINAL;VAR new: PROCESS)

Aquí "P" denota al procedimiento sin párametros que constituye el programa de la última corrutina creada. "A" es la dirección del espacio de trabajo necesario para contener las variables locales de la corrutina y para almacenar el estado de la corrutina mientras esta se encuentra suspendida y "n" denota el tamaño de este espacio de trabajo en unidades de memoria. La corrutina se llama a través la variable "new". Una corrutina inicia su ejecución como resultado de un "Transfer" y termina la misma como resultado de otro.

Cuando un proceso es comenzado con un llamado a StartProcess(P,n), un descriptor y un espacio de trabajo para su corrutina asociada con creados.

El siguiente tipo define al descriptor de procesos.

```
TYPE SIGNAL = POINTER TO ProcessDescriptor;
     ProcessDescriptor =
      RECORD
         next : SIGNAL; (* anillo *)
         queue: SIGNAL; (* cola de procesos en espera *)
         cor  : PROCESS; (* corrutina *)
         ready:  BOOLEAN;  (* indica si el proceso esta listo o
                             no para su ejecución *)
     END;
```

El descriptor es insertado en una lista circular, la cual contiene descriptores de todos los procesos creados. Cualquier proceso puede ser alcanzado avanzando a través del anillo. El proceso sucesor es denotado por el campo next del descriptor.

Dado que el número de esos procesos es desconocido, una solución es organizar estos como una lista encadenada. De aqui que una variable "signal" representa la cabeza de lista, y todos los descriptores de procesos contienen un campo encadenando al siguiente proceso esperando por la misma señal. este valor es NIL si no existe ningún proceso.

De la descripción anterior, las funciones de los procedimientos WAIT y SEND son obvias: SEND(s) saca el primer elemento de la lista "s" y transfiere el control de el proceso que la envia a este proceso. WAIT(s) añade el proceso que lo llama al final de la lista "s".

Esto asegura que todos los procesos seran atendidos de acuerdo con el orden en que lleguen, siguiendo una politica de "el primero en llegar es el primero en salir". En principio cualquier proceso que no este esperando por alguna condición puede ser reanudado, esto se determina siguiendo el anillo desde el proceso actual hasta encontrar uno que tenga un valor cierto en el campo "ready".

CAPITULO 3

MANEJADORES DE INTERRUPCIONES

3.1 Multiprocesamiento

Manejador de Interrupciones es el nombre que se da a un proceso encargado de dar servicio a una interrupción determinada, ya sea generada por Software o por Hardware.

Dado que la arquitectura de la IBM PC es tal que los periféricos tienen sus propios procesadores, es posible tener una autentica concurrencia en nuestro sistema. Para obtener una autèntica concurrencia se asigna un proceso a cada procesador de E/S, el cual realize las acciones adecuadas para tratar la interrupción en cuestión. Esta forma de programar es conocida como multiprocesamiento dado que todos los procesos comparten una memoria común.

Para satisfacer los objetivos de la tésis solo fue necesario instrumentar los siguientes tres manejadores de interrupciones.

- Reloj
- Teclado
- Puerto serie primario

Al reloj le corresponde el procesador 8253
Al teclado el 8048
Al puerto el 8250

3.2 Mecanismo de interrupción en la PC

El procesador 8086/8088 puede reconocer hasta 256 tipos de interrupciones, cada interrupción se especifica por un numero de tipo el cual va desde 0 hasta 255.

Una tabla de vectores de interrupción que contiene hasta 256 vectores de estos es usada por el 8086/8088 para encontrar la dirección de la rutina que da servicio a la interupción ocurrida.

Cada vector de interrupción es un apuntador de dos palabras el cual contine el OFFSET y SEGMENTO de la dirección de la rutina de servicio a la interrpción(RSI).

La fig.3.1 muestra esta tabla

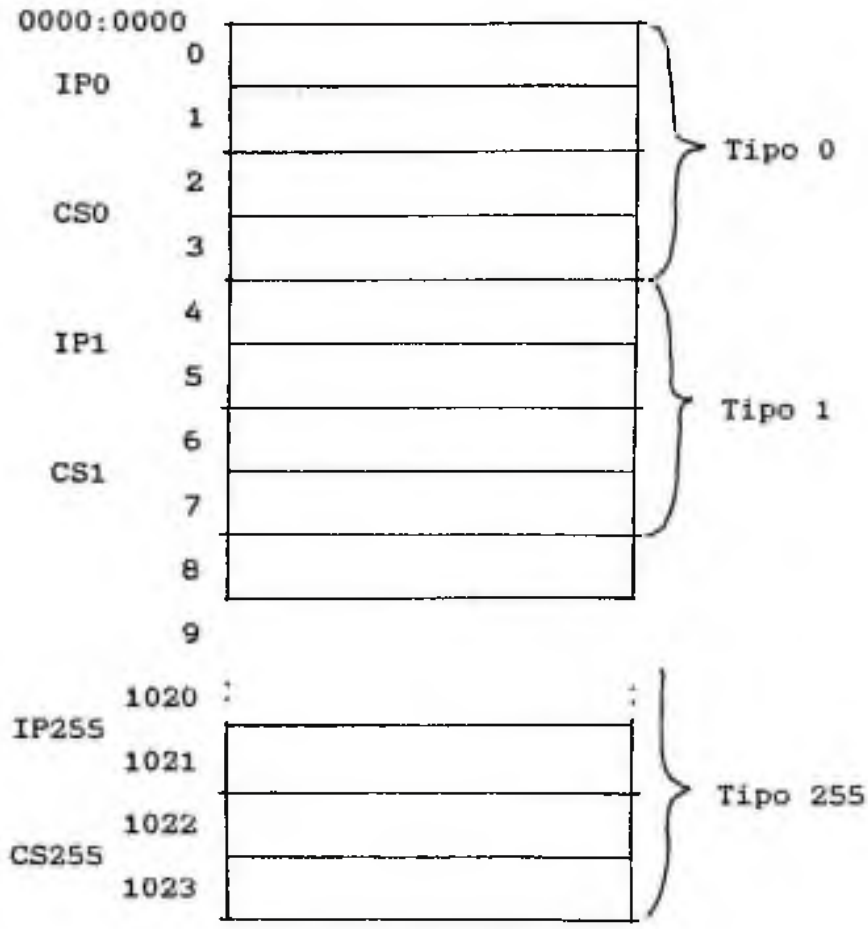


Fig. 3.1 TABLA DE VECTORES DE INTERRUPCION

La tabla comienza en la dirección 0000:0000 y tiene 1K de tamaño(256 vectores, cuatro bytes por vector).

Cuando una interrupción ocurre, el contenido actual del segmento de código (SC) , del apuntador de instrucciones (AP) y de la palabra de banderas son metidos en el Stack(en ese orden). Después el AP y el SC se cargan tomados de la tabla de vectores de interrupción, y la bandera de interrupciones(BI) y el TF(trap-single step-flag) son limpiados.

Las interrupciones externas pueden ser habilitadas en la RSI con la instrucción STI(set interrupt enable flag). La RSI no deberá cambiar ningún registro. Si lo hace el programa interrumpido resultara con errores. Al fin de la RSI, la instrucción IRET restablece la palabra de banderas, IP y CS del Stack [Royer 87].

3.3 Controlador de Interrupciones

Entre el periférico y el 8086/8088 se encuentra el procesador 8259-A, el cual es el controlador de interrupciones. Entre las funciones que tiene están las siguientes:

- Establecer un mecánismo de prioridades para las interrupciones
- Habilitar o deshabilitar interrupciones
- Resolver conflictos sobre que interrupción pasar al procesador 8086/8088.

3.4 Esquema de los Manejadores de Interrupciones

Los manejadores de interrupción usados en este sistema se introducen en el vector de interrupciones como rutina de servicio, estas rutinas no generan cambio de contexto con cada interrupción.

El esquema general de un manejador de interrupciones usado en este trabajo es el dado por el módulo "InterrupHandler", cuyo pseudo-codigo se lista a continuación.

```
IMPLEMENTATION MODULE INTERRUPHANDLER;
 CONST
   No del vector de interrupciones

PROCEDURE DeviceDriver;
BEGIN
 Guardar registros
 habilitar interrupciones

 Cuerpo del proceso

 Enviar fin de atención a la interrupción al Procesador
 8259-A
 Restablecer registros
 Actualizar BP
 Ejecutar IRET
END DeviceDriver;

PROCEDURE ComienzaDriver;
BEGIN
  Guardar valor del vector de interrupción a usar
  Guardar estado de la unidad a usar
  Meter la dirección del la rutina de servicio en el vector
  de interrupciones adecuado
  Habilitar interrupciones del dispositivo a usar
END ComienzaDriver;

PROCEDURE ParaDriver;
BEGIN
  Restablecer direccion original del vector de interrupción
  usado.
  Restablecer estado del dispositivo usado(interrupciones
  habilitadas o interrupciones deshabilitadas)
```

```
END ParaDriver;

PROCEDURE InitDriver;
BEGIN
 Inicializa  parámetros para el manejador de dispositivos si
 esto es necesario.
END InitDriver;

BEGIN

END INTERRUPHANDLER.
```

De este pseudo-codigo podemos apreciar los siguientes cuatro puntos:

- Es muy fácil cambiar o añadir un manejador de interrupciones, debido a la modularidad con que estan organizados.

- Dado que solo se exportan los Procedimientos para instalar y para desinstalar un manejador, es fácil emplearlos sin preocuparse por detalles de instrumentación.

- En estos módulos la comunicación se logra a través de variables globales.

- La exclusión mutua de estas variables se logra habilitando o deshabilitando las interrupciones adecuadas.

CAPITULO 4

DISEÑO DEL SISTEMA

4.1 Modelo Conceptual del Sistema

El modelo conceptual del sistema se presenta en la Fig. 4.1. Este modelo se divide en capas, las cuales se clasifican de acuerdo a los servicios que proveen. La interfaz entre los limites de cada capa es fija (i.e. buffers en donde se deben depositar datos) y, por tanto, es posible modificar la forma de operar o los elementos que componen cada una.

El modelo presenta cuatro capas:

- Capa de adquisición de datos
- Capa de comunicaciones
- Capa de captura de datos
- Capa de aplicaciones

Cada una de estas capas provee un servicio a una capa superior. Esta forma de ver al sistema facilita el crecimiento del mismo de acuerdo a las necesidades del usuario.

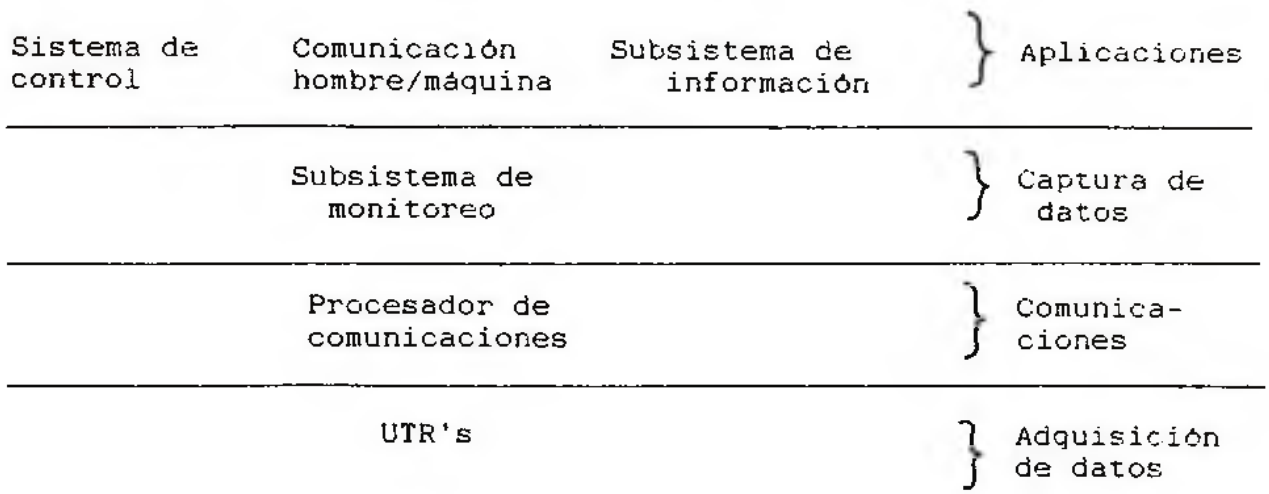


Fig. 4.1 MODELO CONCEPTUAL BASADO EN CUATRO CAPAS DEL SISTEMA

Capa de adquisición de datos:

Tiene como objetivo el recolectar los datos de los puntos de monitoreo, asi como el control de los mismos. Como servicio, la capa superior recibe las lecturas tomadas en esta capa.

Capa de comunicaciones:

Evita un trabajo significativo al computador donde reside el sistema supervisor, pues realiza todo la tarea de verificación de errores y centraliza la información proveniente de la capa anterior. La presente capa se compone de un procesador de comunicaciones y del protocolo que usa para accesar a la capa inferior. El servicio que presta esta capa es mantener información actualizada de los puntos de monitoreo, y entregarla a la capa superior cada vez que se lo solicite.

Capa de captura de datos:

Se encarga de establecer la comunicación entre el computador maestro y el procesador de comunicaciones. La componen los procesos y manejadores de interrupción que tienen que ver con el puerto de comunicaciones, asi como los procesos que interpretan la información recibida. Es aqui donde se procesan los mensajes del procesador de comunicaciones y se determina si existe o no motivo de alarma. También, se encarga de formar los mensajes que provengan del operador del sistema. Esto se hace a través del manejador de interrupciones del teclado. La interfaz entre esta capa y la capa inferior es el puerto de comunicaciones utilizado. El servicio que presta es el envio e interpretación de mensajes.

Capa de aplicaciones:

Se usa la información procesada en la capa inferior con varios fines, por ejemplo, las de llevar estadisticas, obtener reportes, activar a un módulo de control, mostrar al operador la información por medio del video, etc.

4.2 Procesos que componen el Sistema

El término "buffer" no pertenece a la lengua española, pero dado que es un término muy común en el ambiente de computación, se usará en el presente trabajo como sinonimo de "cualquier persona o cosa que sirve para disminuir un choque". Un "buffer" se usa para desacoplar dos procesos de tal manera que estos se mantienen funcionando suavemente. El uso de "buffers" quedará más claro conforme se avanze en la lectura de este capitulo.

De acuerdo con el diseño del sistema en la figura 4.2 vemos que necesitamos los siguientes procesos, manejadores de interrupciones y estructuras de datos: Buffer general, manejador de interrupciones del reloj, del puerto y del teclado, proceso maestro, menupri, interpretamenpc y verificarespc.

- Buffer general:

Este buffer se usa para almacenar mensajes que deben ser atendidos por otros procesos, se calendarizan en forma PEPS(primero en entrar, primero en salir). Cuando este buffer se llena, el proceso que intente depositar un mensaje es detenido, y cuando el buffer esta vacio, el proceso que intente sacar un mensaje será detenido.

- Manejador de interrupciones del teclado:

Este manejador se encarga de formar los mensajes que lleguen del teclado, y después depositarlos en el buffer general. Este manejador se activa con cada tecla que oprime el operador del sistema. Un mensaje se forma caracter por caracter hasta que se recibe un ‹cr›.

- Manejador de interrupciones del reloj:

Este manejador se encarga de examinar las listas de mensajes periódicos y las listas de mensajes no periódicos con el objeto de determinar si ya es tiempo de darles el tratamiento indicado(enviarlos por el puerto, mostrarlos en pantalla ó depositarlos en el buffer general). También verifica si hay o no mensajes en el buffer del puerto, si existen mensajes, entonces se deposita un mensaje en el buffer general, destinado a un proceso que atenderá estos mensajes.

- Manejador de interrupciones del puerto:

La función de este manejador es recibir los caracteres que lleguen por el puerto serie, y formar los mensajes. Estos mensajes se guardan en un buffer circular(BMP), cada vez que se forma un mensaje, se actualiza una variable que me indica el número de mensajes existentes en este buffer(BMP).

- Proceso maestro:

Este proceso se encarga de sacar los mensajes del buffer

general y activar al proceso que dará servicio a este mensaje.
- Proceso menupri:

Este proceso es el encargado de interactuar con el operador, y le presenta las opciones necesarias para la creación y administración de mensajes. También, determina el tiempo que se debe esperar antes de verificar la respuesta a un mensaje enviado al procesador de comunicaciones.

- Proceso interpretamenpc:

Este proceso tiene como función el interpretar los mensajes recibidos por el manejador de interrupciones del puerto, asi como determinar si existe ó no motivo de alarma. Otra función de este manejador es representar en forma gráfica la interpretación del mensaje.

- Proceso verificarespc:

Este proceso se encarga de verificar si ya se recibió o no respuesta a un mensaje enviado al procesador de comunicaciones. en caso de que no sea asi, se activa una alarma indicando esto. El mensaje que activa este proceso se programa al mismo tiempo que el mensaje con destino al procesador de comunicaciones.

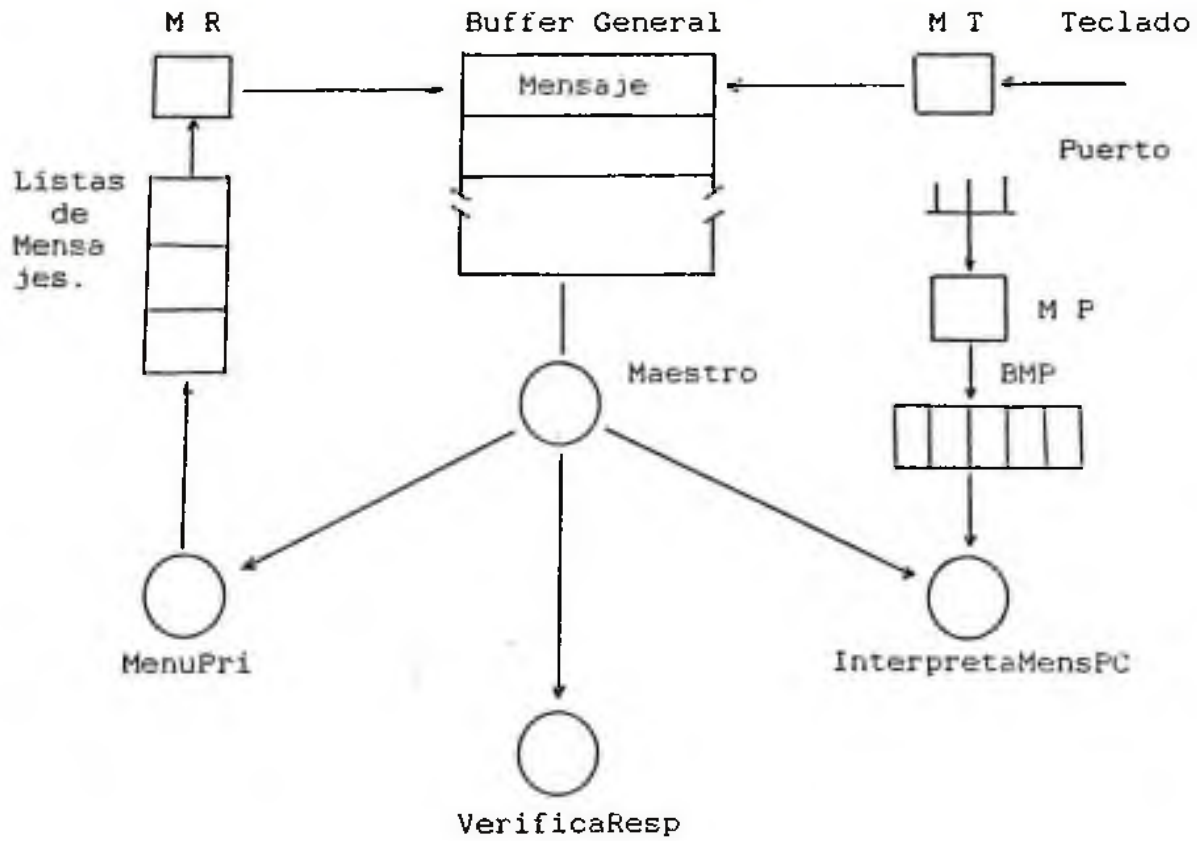


Fig. 4.2 DISEnO GENERAL DEL SISTEMA SUPERVISOR

Donde las siglas tienen los siguientes significados:

- M R .- Manejador del Reloj
- M P .- Manejador del Puerto
- M T .- Manejador del Teclado
- BMP .- Buffer de Mensajes del Puerto

4.3 Activación de Procesos

Los procesos están diseñados en forma tal que no abandonen al procesador hasta terminar su trabajo (excepto si acontece una interrupción) de modo que no se necesita el uso de "rebanadas de tiempo"(time slicing).

Un mensaje es una unidad mínima de información para que un proceso trabaje(excepto en los procesos manejadores de interrupciones). Ejemplos de mensajes son: una pregunta del operador al sistema, una respuesta del sistema al operador, una respuesta de una unidad remota al supervisor, etc.

La relación mensaje - proceso es como sigue:

- Los procesos se activan con mensajes.

- Cuando no hay mensajes pendientes de atención, todos los procesos duermen(excepto un proceso ocioso).

- Cuando llega un mensaje, solo un proceso se hace cargo de éste, cada proceso trabaja con un sólo tipo de mensaje.

Los procesos se activarán de la siguiente manera:

- Usamos un buffer el cual se llamará 'buffer general

- En este buffer general se depositarán todos los mensajes generados por los manejadores de dispositivos para su atención por parte de los procesos de servicio a los mismos.

- Cada mensaje lleva asociado el origen del mismo

- Existe un proceso llamado 'maestro', que dormita mientras no existan mensajes en el "buffer" general.

- Cuando se deposita un mensaje en el buffer general, el proceso maestro se despierta(activa) y saca el mensaje.

- Cuando el proceso maestro saca un mensaje del buffer general, determina el origen del mismo y después despierta al proceso que dará servicio a este mensaje.

- Después del paso anterior el proceso maestro se queda

dormido.

4.4 Buffer General.

Se le llama general porque todos los procesos pueden depositar y sacar mensajes en el. El "buffer" es la principal estructura de datos compartida; junto con las operaciones DEPOSITA y SACA son encapsuladas en un monitor[Wirth 82].

Los procedimientos DEPOSITA y SACA se definen como sigue:

```
Procedure DEPOSITA(x:mensaje);
BEGIN
  n := n+1;
  IF n > N THEN ESPERA(NOLLENO) END;
  (* n < = N *)
  buff[in] := x;
  in := (in + 1) MOD N (* N es el tamaño del buffer *)
  IF n < = 0 THEN ENVIA(NOVACIO) END;
END DEPOSITA;

Procedure SACA(VAR x:mensaje);
BEGIN
  n := n - 1;
  IF  n < 0 THEN ESPERA(NOVACIO) END;
  (* n > = 0 *)
  x := buff[out];
  out := (out + 1) MOD N;
  IF n > = N THEN ENVIA(NOLLENO) END;
END SACA;
```

Este algoritmo es una versión mejorada del 'Barbero Dormilón' realizada por Dijkstra. Este algoritmo extiende el rango de n (* número de elementos depositados *) con el siguiente significado:

```
n < 0 :  El buffer esta vacio y -n consumidores
         esperan.
0 <= n <= N :  n lugares del buffer se están llenando
               no hay procesos esperando.
N < n : El buffer ésta lleno y n-N procesos esperan.
```

Aqúi vemos que todo proceso que intente sacar un mensaje estando el buffer vacio quedará en estado de espera hasta que alguien deposite al menos un mensaje en el buffer.

4.5 Manejadores de Interrupciones

Ahora se verá como se instrumentarón los manejadores de dispositivos.

Como se dijo en 3.1 se tienen tres manejadores de dispositivos instrumentados, estos son:

- Manejador del teclado (MT)

- Manejador del reloj (MR)

- Manejador del puerto (MP)

El sistema fué diseñado de manera que permite crear muy fácilmente manejadores de dispositivos y añadirlos al mismo.

Veamos las funciones de cada uno de ellos.

4.5.1 Manejador del Teclado:

Cada vez que ocurre una interrupción del teclado entra esta rutina a trabajar.

1) Lo primero que se hace es llamar a KB_INT , una rutina del BIOS la cual se encarga de :

- Procesar las 'SHIFT KEYS'(teclas que no generan caracteres alfanumericos p.e. ‹Cntl›, ‹ALT›, ‹Num Lock›, etc.)

- Obtener el código ASCII del carácter tecleado.

- Depositar el carácter obtenido en un buffer llamado "KB_BUFFER".

- Enviar reconocimiento de interrupción al procesador del teclado(8048).

2) Luego se toma el código ASCII depositado en el buffer del teclado (KB_BUFFER) se ejecuta el siguiente algoritmo:

```
IF no es carácter nulo THEN
     CASE dato leido OF
       2: Borrar zona de alarmas
       3: Borrar zona de mensajes
       8: Borrar carácter a la izquierda
      13: Depositar mensaje formado en el buffer
          general
      27: Borrar mensaje que se está formando
      ELSE
        Añadir Dato leido al mensaje en formación
      END (* case *)
  END (* if *)
```

donde

- "dato" es el carácter presionado en el teclado
- Un ‹CR› se considera fin de mensaje y por tanto el mensaje formado se deposita en el buffer general.
- Un ‹ESC› se considera anulación del mensaje
- Cada combinación de ‹Cntl›‹letra› produce un código ASCII que se almacena en la variable dato. Como la variable dato se usa como llave en la instrucción Case, es posible definir acciones para cada combinación de éstas.

4.5.2 Manejador del Puerto.

El manejador del puerto es una rutina de atención a las interrupciones generadas cada vez que llega un caracter al puerto serie RS-232.

La función de este manejador es recibir y formar los mensajes enviados por el dispositivo enlazado al sistema supervisor via RS-232.

La configuración del puerto RS-232 (primario) es la siguiente:

- 2 bits de parada
- Sin paridad
- 8 bits de datos
- Velocidad de transmisión a 9600 bps

De acuerdo con lo anterior vemos que no es conveniente diferir la adquisición de un caracter del puerto puesto que llegan a razón de 1200 caracteres en un segundo.

Para evitar pérdida de caracteres se le dió máxima prioridad a las interrupciones del puerto RS-232 (primario) y el código que se ejecuta en este manejador es muy corto.

El cambio de prioridad se realiza programando el controlador de interrupciones 8259-A realizando una rotación de prioridades enviando un C3H al puerto 20H, la programación del procesador 8259-A para la rotación de prioridades se encuentra en [Intel 82]. Una vez rotadas las prioridades, quedan como se muestra en la fig. 4.3.

0	1	2	3	4	5	6	7
IRQ4	IRQ5	IRQ6	IRQ7	IRQ0	IRQ1	IRQ2	IRQ3

Fig.4.3 CONFIGURACION DE PRIORIDADES EN EL 8259-A

De tal modo que IRQ4 que es el tipo de interrupción que se genera en el puerto serie primario queda con máxima prioridad, e IRQ3 (tipo de interrupción del puerto serie secundario) quedó con la prioridad más baja.

Esta rotación de prioridades se realiza al momento de inicializar el puerto y se mantiene mientras esté activo el supervisor.

El algoritmo que se ejecuta cada vez que llega un carácter al puerto RS-232 es el siguiente:

- Toma el carácter del puerto

- Mete caracter al BufferDeMensajes

- Incrementa apuntador in

- Si se recibe fin de mensaje entonces incrementa número de mensajes recibidos

El bufferDeMensajes es un buffer de 4K (i.e cuatro mil caracteres) instrumentado en forma circular con los apuntadores in y out, el primero apunta a la próxima localidad donde se insertará el caracter y el segundo a la próxima localidad de donde se sacará el caracter.

Inicialmente el numero de mensajes recibidos es 0.

En este módulo se proporciona también, la rutina InitPort, la cual permite modificar la programación del puerto. Esta rutina es muy útil sobre todo si se desea usar el supervisor con otras interfaces más lentas, por ejemplo vía telefónica.

Cuando se satura el bufferDeMensajes no se hace caso a posteriores interrupciones, hasta que al menos se tome un mensaje de éste.

4.5.3 Manejador de Reloj.

Este manejador realiza varias funciones:

- Actualiza fecha y hora.

- Si hay mensajes del procesador de comunicaciones deposita un mensaje en el buffer general dirigido al proceso InterpretaMensPC indicando que hay mensajes.

- Revisa las listas periódicas y las no periódicas, si se cumple su tiempo se envia el mensaje indicado.

Estas acciones se realizan cada segundo.

Durante la ejecución de este manejador las interrupciones permanecen habilitadas, de tal forma que, toda interrupción con prioridad mayor será ejecutada excepto en zonas criticas. Se considera zona critica cuando se accesan datos compartidos con otros manejadores de interrupción.

Para evitar que interrumpan al manejador durante la ejecución de una zona critica, se inhabilitan las interrupciones del manejador que afecte a esta zona critica directamente

El algoritmo que realiza el manejador del reloj es el siguiente:

```
Si NumDeTicks = segundo Entonces
  Actualiza fecha y hora
  Si NumDeMens <> 0 Entonces DEPOSITA(MensDePuerto)
END;
```

Si tenemos un mensaje periódico a enviar en este tiempo Entonces EnviaAPuerto ó DEPOSITA(mensaje) ó escribe mensaje en pantalla END;

Si tenemos mensaje no periódico a enviar en este tiempo Entonces EnviaAPuerto ó DEPOSITA(mensaje) ó escribe mensaje en pantalla y después borralo de la lista de mensajes no periódicos END;

- NumDeTicks es un contador que me indica cuando se ha cumplido un segundo. Un segundo equivale a 18 ticks de reloj.

- NumDeMens es una variable que me indica si hay mensajes en el buffer del puerto. Esta variable es actualizada por el manejador del puerto.

- DEPOSITA es el procedimiento que se encarga de meter un mensaje al buffer general.
- EnviaAPuerto es un procedimiento que se encarga de enviar un mensaje por el puerto serie RS-232.

Son cuatro tipos de mensajes que se manejan:

- Mensajes al procesador de comunicaciones.

- Mensajes personales, es decir hacia la pantalla.

- Mensajes al proceso "VerificaRespuestaPC"

- Mensajes al proceso "InterpretaMensajePC"

donde, los mensajes dirigidos al puerto, a pantalla y al proceso "VerificaRespuestaPC" pueden ser periódicos ó no periódicos(en el inciso 4.6.2.1 se vera la diferencia de unos y otros).

4.6 Procesos creados con el núcleo de concurrencia

Son cuatro los procesos creados con este núcleo, aparte del proceso ocioso creado al inicializarse el módulo en que se instrumenta este núcleo. Estos son:

- Proceso Maestro

- Proceso MenuPri

- Proceso InterpretaMensajePC

- Proceso VerificaRespuestaPC

Cada proceso atiende a un tipo de mensaje específico y el código de ejecución en cada proceso es bastante corto y rápido.

4.6.1 Proceso Maestro

Este proceso se encarga de despertar al proceso indicado de acuerdo con el mensaje que sacó del buffer general. Por la función que cumple de repartir el trabajo entre los diferentes procesos también se llamara el despertador de procesos o asignador de tareas.

Cuando el proceso es creado, trata de sacar un mensaje del buffer general. Si no hay mensajes, se duerme y cuando alguien deposita un mensaje se despierta este proceso.

Cuando el proceso maestro despierta, lo primero que hace es sacar el mensaje del buffer general y ver el remitente. Le quita la identificación y despierta al destinatario, el cual conoce de acuerdo al remitente. Al mismo tiempo deposita el mensaje en una variable global de

tal suerte que lo primero que hace el proceso despertado es ver en esa variable el mensaje que causo que lo despertaran. En este instante el proceso maestro está durmiendo hasta que el proceso actual termina su trabajo.

Si el proceso maestro recibe un '*', entonces termina el trabajo del supervisor.

4.6.2 Proceso MenuPri.

Este proceso es el encargado de atender los requerimientos del operador del sistema, de tal manera que, siempre que se requiera un dato de consola, el proceso se pone a dormir, y cuando es despertado por el maestro es porque ya tiene lo necesario para trabajar.

Por el momento, el usuario del sistema supervisor solo tiene interacción con el sistema administrando las listas de mensajes, para lo cual se presenta un menú con las siguientes opciones principales:

- Insertar mensaje periódico
- Insertar mensaje no periódico
- Borrar mensaje periódico
- Borrar mensaje no periódico
- Listar mensajes periódicos
- Listar mensajes no periódicos
- Listar un tiempo periódico
- Listar un tiempo no periódico

El manejo de mensajes se realiza a través del módulo listas. Este módulo es el encargado de crear, borrar y buscar listas periódicas y no periódicas asi como el insertar, borrar y buscar mensajes dentro de esas listas. Estas listas están doblemente ligadas y tienen una "Cabecera"

4.6.2.1 Tipos de Mensajes

Un mensaje periódico es aquél que se envia en forma ciclica cada cierta cantidad(programable) de segundos.

Un mensaje no periódico es aquel que se envia solo una vez y después se borra de la lista.

Debido a las caracteristicas del sistema, se determinó un periodo minimo de un segundo entre mensajes. Ya que entre que el PC pregunta a una UTR y recibe respuesta, transcurre mas de un segundo.

Una vez insertado un mensaje periódico en la lista correspondiente, solo se borrará mediante orden directa del operador.

Para insertar un mensaje no periódico se necesita dar la fecha y hora en que lo debe tomar el proceso encargado de despacharlo.

Existen tres tipos de mensajes que pueden ser periódicos ó no periódicos.

- Mensaje personal
- Mensaje a procesador de comunicaciones
- Mensaje para verificar respuesta de PC

- Mensaje personal:

Estos mensajes son de tipo texto y tendrán una longitud máxima de ochenta caracteres. Una vez que se cumple el tiempo de despacho del mensaje, éste se imprime en el area de mensajes personales, la cual se encuentra en la fila 24 de la pantalla del operador.

- Mensaje a procesador de comunicaciones:

Este tipo de mensajes se forma automáticamente, guiando al operador a través de menús con el fin de llenar los parámetros necesarios para formar el mensaje. Esto se hace de la siguiente manera:

- Que tipo de mensaje ?
  1) Personal
  2) A Procesador de comunicaciones

si el mensaje es personal pregunta el texto del mismo.

Si el mensaje es dirigido al PC entonces se pregunta por el tipo de mensaje a PC que se desea enviar. Estos pueden ser:

- 1). dame valor de todas las variables
- 2). dame estado de UTR
- 3). dame valor de variable X.
  : :

Dependiendo del tipo de mensaje, se pregunta por los parámetros necesarios.

Una vez que se termino de formar el mensaje, se pide el tiempo ó la fecha y la hora en que será procesado el mensaje.

- Mensaje para verificar respuesta del procesador de comunicaciones.

Hay varias causas por las que un mensaje determinado puede variar su tiempo de respuesta. Entre estas causas están la distancia a que se encuentra la UTR del procesador de comunicaciones, la complejidad del mensaje, etc. Por tanto, no es conveniente establecer un tiempo fijo antes de verificar si ya se recibió respuesta a un mensaje determinado ó no.

Debido a lo anterior, cada vez que el operador programa un mensaje dirigido al procesador de comunicaciones, automáticamente se programa un mensaje con el tiempo que se debe esperar antes de verificar la respuesta por parte del procesador de comunicaciones. El tiempo que se debe esperar se determina por el tipo de mensaje programado.

Para borrar un mensaje se siguen los mismos pasos que para insertarlo. Al momento de borrar un mensaje tipo pregunta al PC, también se borra en forma automática el mensaje tipo verifica respuesta correspondiente al mismo.

El diseño de este proceso permite la adición de tipos de mensajes de una manera fácil y eficiente.

4.6.3 Proceso VerificaRespPC

Este proceso es el encargado de determinar si se recibio ó no la respuesta del procesador de comunicaciones a un cierto mensaje tipo pregunta.

Este proceso y el proceso "InterpretaMensajePC" usan la tabla "CodigosRecibidos", que se muestra en la fig. 4.4.

0	Canal	UTR
1	x	y
: n	:	:

Fig. 4.4 TABLA "CODIGOSRECIBIDOS"

donde :

- n es el tipo de pregunta enviada al PC
- x es el número de canal por el que se envió el mensaje.
- y es el número de la UTR destino.

El proceso "InterpretaMensajePC" incrementa el canal y UTR correspondiente al tipo de respuesta recibido.

Cuando el proceso maestro recibe un mensaje tipo VerificaRespuesta, despierta a este proceso.

Un mensaje tipo VerificaRespuesta contiene la siguiente información:

- x tipo de mensaje a verificar
- y número de canal a verificar
- z número de UTR a verificar

con esta información lo que hace este proceso es contestar la pregunta:

- Recibí un mensaje tipo x por el canal y de la UTR z?

si la respuesta es "NO", entonces envia al operador del sistema un mensaje de alarma indicando el tipo de mensaje, el número de canal y el número de la URT delincuente.

La manera en que el proceso contesta la pregunta anterior es verificando que el Canal y UTR asociado al tipo n no sean cero. Si son cero determina que la respuesta es "NO". Si no son cero decrementa sus valores en uno.

De esta manera se asegura que se checarán tantos mensajes de un canal, tipo y UTR como sean enviados al PC.

Es importante conocer con certeza el tiempo de respuesta de cada dispositivo, y dejar un margen de seguridad adecuado.

4.6.4 Proceso InterpretaMensajePC

Este proceso es el encargado de interpretar los mensajes enviados por el procesador de comunicaciones al sistema supervisor.

El sistema supervisor está conectado a un procesador de comunicaciones, y este a su vez está enlazado vía telefónica con dos UTR's.

Cada UTR tiene ocho variables digitales controladas por interruptores de dos estados, 1 ó 0.

Cada vez que se envia un mensaje al procesador de comunicaciones, se recibe una respuesta de él mismo con este formato:

- syn
- syn
- ACK o NAK
- Código de error
- Código de operación
- No de variables digitales
- No de variables analógicas
- No de canal
- No de UTR
- No de variable digital
- Valor de variable

  :

- No de variable analógica
- Valor de variable

  :

- eot
- eot

Este proceso, 'InterpretaMensajePC', lee el mensaje. Si hay un NAK, envia una alarma al operador mostrando el tipo de error indicado en el código de error del mensaje. Si el mensaje es correcto, incrementa los valores correspondientes en la tabla de CodigosRecibidos.

Si el mensaje viene sin error, entonces determina el significado de las variables leídas, y representa en pantalla la interpretación del mensaje recibido. Si la

interpretación del mensaje determina que existe motivo de alarma, entonces se envía el mensaje adecuado al operador del sistema.

4.7 Flujo de Datos del Procesador de Comunicaciones al Proceso "InterpretaMensajePC"

Se hacen nueve pasos:

- Se envia pregunta al procesador de comunicaciones.

- El manejador del puerto recibe la respuesta e incrementa la variable de número de mensajes recibidos "NumDMen".

- El manejador de reloj, pregunta cada segundo si el número de mensajes recibidos por el puerto es diferente de cero.

- Si es así, entonces deposita un mensaje en el buffer general dirigido al proceso "InterpretaMensajePC".

- El proceso maestro recibe el mensaje y despierta al proceso InterpretaMensajePC.

- El proceso InterpretaMensajePC antes que nada pregunta cuantos mensajes se encuentran en el buffer de mensajes del puerto.

- Una vez que sabe èsto, decrementa el contador de número de mensajes deshabilitando en esta instrucción las interrupciones.

- Interpreta tantos mensajes como existían cuando fuè despertado(activado).

- El proceso InterpretaMensajePC se queda dormido y activa al proximo proceso listo.

4.8 Ejemplo para el Sistema Supervisor

Para probar el supervisor supondremos lo siguiente:

- Cada UTR nos envia ocho variables digitales, las combinaciones de estas nos indican diferentes cosas.

- Supondremos que cada UTR es un elevador.

- Entonces tendremos dos elevadores a supervisar.

Los significados de las combinaciones de las diferentes variables son las siguientes:

1 2 3 4	Piso
0 0 0 0	1
0 0 0 1	6
0 0 1 0	2
0 0 1 1	7
0 1 0 0	3
0 1 0 1	8
0 1 1 0	4
0 1 1 1	9
1 0 0 0	5
x x x x	Anterior

5 6	Estado
0 0	Anterior
0 1	Parado
1 0	Subiendo
1 1	Bajando

7	Puerta
0	Abierta
1	Cerrada

8	Alarma
0	no
1	si

En este caso :
el piso 6 es entre 1 y 2
el piso 7 es entre 2 y 3
el piso 8 es entre 3 y 4
el piso 9 es entre 4 y 5

x es no importa el valor

El proceso InterpretaMensajePC está hecho para decidir las acciones a tomar en base a la interpretación hecha de las variables recibidas del elevador.

Las tareas que realiza este proceso son:

- Checar que el mensaje llegue sin error

- Determinar el :
  - Piso en que se encuentra.
  - Estado del elevador.
  - Estado de la puerta.
  - Estado de la alarma interna del elevador.

- Determinar condiciones de alarma.

- Representar gráficamente los resultados de las interpretaciones.

- Avisar al operador en caso de alarma.

- Actualizar la tabla de códigos recibidos.

Las condiciones de alarma son las siguientes:

- Elevador xx subiendo y puerta abierta.
- Elevador xx bajando y puerta abierta.
- Elevador xx parado entre dos pisos.
- Elevador xx Puerta abierta entre pisos.
- Elevador xx pasajero con problemas.

Donde xx es la referencia del elevador con problemas

Si ponemos las variables del las UTR's de tal modo que el elevador esta en el piso 5 y el estado del elevador es subiendo, no se realiza ninguna acción. Igual tratamiento se da cuando el elevador está en el piso número uno y el estado del mismo es bajando.

CAPITULO 5

CONCLUSIONES:

A través del desarrollo del presente trabajo se han cubierto los siguientes temas:

- Concurrencia
- Interrupciones
- Tiempo real
- Supervisores

Uniendo estos conceptos ha surgido el sistema supervisor de procesos aquí expuesto.

El sistema operativo MSDOS no es reentrante(i.e. cuando alguna de sus funciones se interrumpe, al reanudarse, puede estar en estado inconsistente y provocar falla del sistema). Las operaciones de Entrada/Salida provistas por Modula-2 hacen uso de las funciones de MSDOS. Debido a esto, fué necesario instrumentar rutinas propias de Entrada/Salida. Por el mismo motivo, si se desea usar disco, será necesario programar un manejador de archivos el cual no use las rutinas del MSDOS para el acceso a los mismos(esto último se encuentra fuera del alcance de esta tesis).

El Sistema Supervisor esta diseñado de tal modo que permite aumentar su complejidad con pocos cambios, y dado el enfoque modular que prevalece en el sistema esta tarea se facilita en forma significativa.

Sobre el lenguaje utilizado Modula-2 se puede decir que:

- Proporciona las facilidades necesarias para realizar este tipo de sistemas pues es fácil de accesar la programación a bajo nivel.
- Contempla el sistema de programación modular, permitiendo el uso de monitores.
- Las primitivas TRANSFER e IOTRANSFER forman parte del lenguaje lo cual permite la validación de las mismas por parte del compilador.
- Teniendo estas primitivas es relativamente fácil instrumentar un kernel ó núcleo el cual instrumente las políticas de administración de procesos que se deseen.

El sistema supervisor ha sido probado en computadoras personales del tipo XT, ST y AT con velocidades de cuatro y ocho Mhz. El desempeño del sistema mejora, obviamente, con la velocidad de procesador usado.

CAPITULO 6

REFERENCIAS

[Andrews 1983].Andrews and Schnèider, Concepts and Notations for Concurrent Programing. Computing Surveys, Vol. 15, No 1, March 1983.

[Dijkstra 68] Dijkstra, E. W., The structure of the "THE" multiprograming system, Commun. ACM 11, 5(May 1968), pp 341-346.

[Gerez 1984] Victor Gerez, Mauricio Mier, Rolando Nieva y Guillermo Rodriguez, Desarrollo y Administración de Programas de Computadora, CECSA, pp 87 - 84, 1984.

[Intel 1982] Intel, Componet Data Catalog, pp 7-129, 7-130. 1982.

[Knuth 1973] Donald E. Knuth, Algoritmos Fundamentales, Ed. Everest, S.A. , pp 205 - 207, 1973.

[Royer 1987] Jeffrey P. Royer, Handbook of software and hardware interfacing for IBM PCs. Prentice Hall, Englewood Cliffs, New Jersey 07632. 1987.

[Young 1982] Stephen J. Young, Real Time Languages: Design and Development, Ellis Horwood Limited, pp 246 - 279, 1982.

[Wirth 1977a] Niclaus Wirth,"Modula: A Lenguage for modular multiprograming."Softw. Pract. Exper., 7 (1977), 3 - 35.

[Wirth 1982] Niclaus Wirth,"Programming in modula-2" Springer Verlag, New York, 1982.

[Logitech 1984] Logitech, Inc. "manual of Modula-2/86" 805 Veterans Blvd. Redwood City, CA 94063 USA.

APENDICE A

Modula-2

Modula-2 es un descendiente directo de Pascal[Wirth 71] y Modula[Wirth 77].

Tras un cuidadoso diseño, Modula-2 surgió como un lenguaje que incluye todos los aspectos de Pascal y extiende estos con los conceptos de módulo y multiprogramación provistos por Modula.

Un módulo es un conjunto de procedimientos, variables, constantes, y tipos de datos, el cual a su vez se puede componer de otros modulos.

Un módulo puede hacer referencia a un objeto(procedimiento, variable, constante, tipo de dato) definido en otro módulo. Esto siempre y cuando el módulo que define los objetos los tenga explicitamente en una lista de objetos a exportar, y el módulo que hace referencia a estos los tenga en una lista de objetos a importar.

A su vez, un módulo que exporta objetos se compone de un módulo de definición y un módulo de instrumentación. En el módulo de definición se realiza la declaración de los objetos exportados, y en el módulo de instrumentación la codificación referente a estos objetos(p.e. el encabezado de un procedimiento que se exporta, se encuentra en el módulo de definición y el cuerpo del procedimiento se encuentra en el módulo de instrumentación.)

Esto permite que se cambie el cuerpo del procedimiento sin cambiar la parte de definición del mismo, y por tanto sin necesidad de alterar los módulos que lo importan.

Debido a que su sintaxis es más parecida a la de Modula que a la de Pascal, se le llamó Modula-2.

Un objetivo de Modula-2 es atacar aquellas areas que tradicionalmente dominaba la programación en lenguaje ensamblador. Es por tanto útil para programar pequeños sistemas basados en microprocesadores, y ha encontrado aplicación en los sistemas relacionados con programación en tiempo real, como son control de procesos y supervisores.

Las diferencias más significativas de Modula-2 con respecto a Pascal[Young 82] son las siguientes:

- La introducción del concepto de Módulo, y en particular la capacidad de dividir un módulo en dos partes: Definición e Intrumentación.

- Una sintaxis mas sistemática que facilita su aprendizaje. En particular todas las estructuras que comienzan con una palabra clave también terminan con una palabra clave.

- El concepto de proceso como la clave para efectuar multiprogramaciòn.

- Las capacidad de efectuar programaciòn en bajo nivel.

- El tipo PROCEDURE, el cual permite que los procedimientos puedan ser asignados dinamicamente a variables.

Los comentarios encontrados en la implementaciòn de modula-2 [Logitech, 84] que se uso para el desarrollo de esta tesis,fueron de gran utilidad, sobre todo lo referente a facilidades de bajo nivel.

APENDICE B

LISTADOS FUENTE

```
DEFINITION MODULE Procesos;
(*  exporta las estructuras de datos y procedimientos  necesarios
para  el uso de senales.  Administra los procesos que esperan por
una senal *)

  EXPORT QUALIFIED
    SENAL,ENVIA, ESPERA, CreaProceso, Inicializa, AlguienEspera;

  TYPE SENAL;

  PROCEDURE CreaProceso(P         :PROC;
                        WsSize    :CARDINAL);

  PROCEDURE ENVIA(VAR SenalMandada: SENAL);

  PROCEDURE ESPERA(VAR SenalPorLaQueEspero : SENAL);

  PROCEDURE Inicializa(VAR SenalAInicializar : SENAL);

  PROCEDURE AlguienEspera( SenalAChecar : SENAL):BOOLEAN;


END Procesos.
```

```
IMPLEMENTATION MODULE Procesos[6] ;
(*
Title    : Procesos
Comment : Este Modulo corresponde al Standart sugerido por N. Wirth
LastEdit: 01 - Febrero - 87
Author   : Felipe Verdalet Guzman
System   : LOGITECH MODULA-2/86
*)
  FROM SYSTEM IMPORT ADDRESS, PROCESS, NEWPROCESS, TRANSFER, TSIZE;

  FROM Storage IMPORT ALLOCATE;
  FROM  Screen IMPORT WriteString,WriteLn;

  TYPE
       SENAL = POINTER TO  DescriptorDeProcesos;
       DescriptorDeProcesos = RECORD
         SiguienteSenal  : SENAL;        (* Circular *)
         ColaDeEspera    : SENAL;        (* cola de procesos esperando *)
         Corrutina       : PROCESS;
         EstadoListo     : BOOLEAN;
       END;

   VAR
       ProcesoActual:SENAL;

   PROCEDURE CreaProceso(P         :PROC;
                         WsSize    :CARDINAL);
      VAR
          ProcesoPrevio : SENAL;
          WsAdr         : ADDRESS;        (* Direccion para espacio de trab.
   BEGIN
     ProcesoPrevio := ProcesoActual;
     ALLOCATE(WsAdr, WsSize);
     ALLOCATE(ProcesoActual, TSIZE(DescriptorDeProcesos));
     WITH ProcesoActual^ DO
       SiguienteSenal                  := ProcesoPrevio^.SiguienteSenal;
       ProcesoPrevio^.SiguienteSenal := ProcesoActual;
       EstadoListo                     := TRUE;
       ColaDeEspera                    := NIL;
     END (* With *);
     NEWPROCESS(P, WsAdr, WsSize, ProcesoActual^.Corrutina);
     TRANSFER(ProcesoPrevio^.Corrutina, ProcesoActual^.Corrutina);
 END CreaProceso;

  PROCEDURE ENVIA( VAR SenalMandada : SENAL) ;

    VAR
        ProcesoQueEnviaSenal : SENAL;
  BEGIN
    IF SenalMandada <> NIL THEN
      ProcesoQueEnviaSenal := ProcesoActual;
      ProcesoActual        := SenalMandada;
      WITH ProcesoActual^ DO
        SenalMandada  := ColaDeEspera;
        EstadoListo   := TRUE;
```

```
          ColaDeEspera  := NIL;
        END (* With *);
        TRANSFER(ProcesoQueEnviaSenal^.Corrutina,
                 ProcesoActual^.Corrutina);
      END (* if *);
    END ENVIA;

    PROCEDURE ESPERA(VAR SenalPorLaQueEspero: SENAL);
      VAR
          EstaSenal       : SENAL;
          SiguienteSenal : SENAL;
    BEGIN (* Inserta el proceso actual en la cola de espera *)
          (* Por la Senal                                    *)
      IF SenalPorLaQueEspero = NIL THEN
        SenalPorLaQueEspero := ProcesoActual;
      ELSE
        EstaSenal           := SenalPorLaQueEspero;
        SiguienteSenal      := EstaSenal^.ColaDeEspera;
        WHILE SiguienteSenal <> NIL DO
          EstaSenal         := SiguienteSenal;
          SiguienteSenal    := EstaSenal^.ColaDeEspera;
        END (* while *);
        EstaSenal^.ColaDeEspera  := ProcesoActual;
      END (* if *);
      EstaSenal := ProcesoActual;
      REPEAT
        ProcesoActual  := ProcesoActual^.SiguienteSenal;
      UNTIL ProcesoActual^.EstadoListo;
      IF ProcesoActual = EstaSenal THEN
        WriteString(" OCURRIO UN ABRAZO MORTAL");
        WriteLn();
        HALT;
      END (* if *);
      EstaSenal^.EstadoListo:= FALSE;
      TRANSFER(EstaSenal^.Corrutina,ProcesoActual^.Corrutina);
    END ESPERA ;

    PROCEDURE AlguienEspera( SenalAChecar: SENAL) : BOOLEAN;
    BEGIN
      RETURN(SenalAChecar <> NIL);
    END AlguienEspera;

    PROCEDURE Inicializa(VAR SenalAInicializar : SENAL) ;
    BEGIN
      SenalAInicializar := NIL;
    END Inicializa;

    BEGIN
      ALLOCATE(ProcesoActual,TSIZE(DescriptorDeProcesos));
      WITH ProcesoActual^ DO
        SiguienteSenal := ProcesoActual;
        EstadoListo    := TRUE;
        ColaDeEspera   := NIL;
      END; (* WITH *)
END Procesos.
```

```
DEFINITION MODULE buffer1;
(* exporta un buffer circular limitado y los procedimientos  para
depositar y sacar mensajes de este *)

EXPORT QUALIFIED DEPOSITA, SACA, mensaje,TamaMens;

CONST
      TamaMens  =  50;  (* tamanio maximo de los mensajes que  se
manejan en este buffer *)

TYPE     mensaje = ARRAY[0..TamaMens] OF CHAR;

PROCEDURE DEPOSITA(x:mensaje);

PROCEDURE SACA( VAR x:mensaje);

END buffer1.
```

```
IMPLEMENTATION MODULE buffer1[6];

(* El buffer se encuentra en un monitor para asegurar la
exclusion mutua. Cuando un proceso accesa un procedimiento,
ningun otro proceso con prioridad igual o menor lo puede accesar
*)

FROM Procesos IMPORT
  SENAL,ENVIA,ESPERA,Inicializa;

FROM Screen IMPORT WriteString;
  CONST
       N = 100; (* Tamano del buffer *)
  VAR
      n:[-1..N+1];          (* # DE ELEMENTOS DEPOSITADOS *)
      NOLLENO:SENAL; (* INDICA QUE n <=  N *)
      NOVACIO:SENAL; (* INDICA QUE n > 0 *)
      in,out:[0..N-1]; (* Indices del buffer *)
      ColaMensajes:ARRAY [0..N-1] OF mensaje;

 PROCEDURE DEPOSITA(x:mensaje);
 BEGIN
   n:= n+1; (* incrementa el numero de elentos depositados *)
   IF n > N THEN ESPERA(NOLLENO) END;
            (* espera  que n < = N para poder depositar *)
   ColaMensajes[in]:= x;
   in := (in + 1) MOD N;
   IF n<= 0 THEN ENVIA(NOVACIO) END;
 END DEPOSITA;

 PROCEDURE SACA(VAR x:mensaje);
 BEGIN
    n:= n - 1; (* Indica que se esta sacando un caracter *)
    IF n < 0 THEN  ESPERA(NOVACIO) END;
    x:= ColaMensajes[out];
    out:=(out + 1) MOD N;
    IF n > N THEN ENVIA(NOLLENO) END;
 END SACA;

 BEGIN
   n := 0;
   in:= 0;
   out:=0;
   Inicializa(NOLLENO);
   Inicializa(NOVACIO);
 END buff
```

```
DEFINITION MODULE TIMER;
  EXPORT QUALIFIED

    ComienzaTimer, ParaTimer;

    (* comienza a funcionar el despachador de listas *)
    PROCEDURE ComienzaTimer;

    (* Deja de funcionar el despachador de listas    *)
    PROCEDURE ParaTimer;

END TIMER.

IMPLEMENTATION MODULE TIMER;
(*
Title    : teclado
Comment : El presente modulo instrumenta despachador de mensajes.
          estos son , mensajes periodicos y mensajes no periodicos,
          cada segundo se revizan las listas y cuando se cumple el
          tiempo especificado en cada lista, los mensajes que se
          encuentren en esta se envian al proceso maestro mendiante
          el buffer principal.
          ---------------------------------------------------
LastEdit: 23 DE JUNIO DE 1987
Author   : Felipe Verdalet Guzman.
System   : LOGITECH MODULA-2/86
*)
 FROM LISTAS IMPORT PrimerPeriodico,PrimerNoPeriodico,TiempoPtr,
                    TiempoNoPerPtr,MensajePtr,EuscaLista1,BorraLista,
                    BorraMensaje1;
 FROM Strings IMPORT
    CompareStr, Delete, Length, Concat;
 FROM buffer1 IMPORT
   DEPOSITA,mensaje, TamaMens;

 FROM utilerias IMPORT criticalSectionPtr,DameHoraSistema,DameFechaSistema;

 FROM SYSTEM IMPORT
    PROCESS, NEWPROCESS, TRANSFER, IOTRANSFER, ADR, SIZE,
    BYTE, ADDRESS, INBYTE,OUTBYTE,ENABLE,DISABLE,LISTEN,CODE;

 FROM System IMPORT
    TermProcedure;

 FROM Devices IMPORT
   GetDeviceStatus, SetDeviceStatus, SaveInterruptVector,
   RestoreInterruptVector;
  FROM Puerto1 IMPORT NumDeMen;
  FROM NumberConversion IMPORT CardToString,IntToString,NumToString;
  FROM Screen IMPORT GotoXY,WriteString;
    CONST
          segundo = 18;
          minuto = 1080;
TYPE
     ScreenType =
```

```
          ARRAY[0..24] OF ·            (* filas *)
          ARRAY[0..79] OF              (* columnas *)
          RECORD char,attr:CHAR END; (* contenido *)
VAR
    screen[0B800H:0H]:ScreenType;
    e:CARDINAL;
    s:ARRAY[0..6] OF CHAR;
    t:ARRAY[0..19] OF CHAR;


CONST                                  (*************************************
     device = 0;                       (* numero de bit en la mascara del
                                       (* controlador de interrupciones.

     InterruptVectorNumber = 1CH;      (* Numero del tipo de interrupcion
                                       (* Tipo de interrupcion generada por
                                         (* el TECLADO
)
VAR

    OldInterruptVector:ADDRESS;        (* guarda un vector de interrupcion


    OldDeviceStatus:BOOLEAN;           (* guarda el estado del dispositivo
                                       (*  (interrupciones habilitadas o no)
    activ:BOOLEAN;
                                       (* indica cuando a sido activado el
                                       (*  manejador de dispositivos
    VerifMensPuerto:mensaje;
    Tick           : CARDINAL;
    Aux            : TiempoPtr;
    Aux1           : TiempoNoPerPtr;
    AuxMen         : MensajePtr;
    Cabeza         : CHAR;
 (*$S-*)
 (*$T-*)
 (*$R-*)
(* Quitamos la prueba del stack porque este es llamado
   despues de salvar los registros, de tal modo que
   se pueden destruir *)

   PROCEDURE WritePuerto(ch:CHAR);
  VAR
      status : CARDINAL;
     T,fantasma : INTEGER;
  BEGIN
    INBYTE(3F8H+5,status);
    LOOP
      IF 5 IN BITSET(status) THEN
        EXIT
      END (* if *);
    END (* loop *);
    OUTBYTE(3F8H,ch);
    FOR T:=0 TO 250 DO
    END (* for *);
  END WritePuerto;
```

```
PROCEDURE DeviceDriver;
BEGIN
     (* prologo al procedimiento generado por el compilador:
        PUSH BP
        SUB SP, espacio variable
 *)
 (* salvamos todos los registros *)
 CODE(50H);   (* PUSH AX *)
 CODE(51H);   (* PUSH CX *)
 CODE(52H);   (* PUSH DX *)
 CODE(53H);   (* PUSH BX *)
 CODE(56H);   (* PUSH SI *)
 CODE(57H);   (* PUSH DI *)
 CODE(1EH);   (* PUSH DS *)
 CODE(06H);   (* PUSH ES *)
 CODE(0FBH);
   INC(Tick);
   IF Tick = segundo THEN          (* Actualiza hora y fecha *)
     Tick := 0;
     INC(SEC);
     IF SEC = 60  THEN
       SEC:=0;INC(MIN);
       IF MIN = 60  THEN
         MIN:=0;INC(HORA);
         IF HORA = 24 THEN
           HORA:=0;INC(DIA);
           IF (DIA = 29) AND (MES = 3)  THEN
              DIA:= 1;INC(MES);
              IF MES = 13 THEN
                MES := 1; INC(ANIO);
              END;
           ELSIF
              (DIA = 31) AND (MES IN (4,6,9,11)) THEN
                DIA := 1; INC(MES);
                IF MES = 13 THEN
                  MES := 1; INC(ANIO);
                END;
           ELSIF
              (DIA = 32) AND (MES IN (1,3,5,7,8,10,11)) THEN
                DIA := 1; INC(MES);
                IF MES = 13 THEN
                  MES := 1; INC(ANIO);
                END;
           END;  (* if dia *)
         END; (* if hora *)
       END; (* if min *)
     END; (* if sec *)
     (***** formar string con fecha y hora 'dd-mm-aaaa   hh:mm:ss' ***)
     NumToString(DIA,10,s,2);
     Concat(t,s,t);
     NumToString(MES,10,s,2);
     Concat(t, '-',t);
     Concat(t,s,t);
     NumToString(ANIO,10,s,4);
```

```
 Concat(t,'-',t);
 Concat(t,s,t);
 Concat(t,'  ',t);
 NumToString(HORA,10,s,2);
 Concat(t,s,t);
 Concat(t,':',t);
 NumToString(MIN,10,s,2);
 Concat(t,s,t);
 Concat(t,':',t);
 NumToString(SEC,10,s,2);
 Concat(t,s,t);
 (***** escribir string en pantalla ****)
 FOR e:=0 TO Length(t)-1 DO
   screen[0,e+50].char:= t[e];
 END (* for *);
 Delete(t,0,20);
 CODE(0FAH);                          (* verifica por mensajes el puerto*
 IF NumDeMen <> 0 THEN
   CODE(0FBH);
   SetDeviceStatus(6,FALSE);
   DEPOSITA(VerifMensPuerto);
   SetDeviceStatus(6,TRUE);
 END;
CODE(0FBH);
 (* Checar listas de tiempos no periodicos para ver si hay
    alguna con la fecha y hora actual, si existe se insertan
    los mensajes que tengan en la cola circular del buffer1
    y despues se borra la lista                                  *)
 IF PrimerNoPeriodico^.Next <> PrimerNoPeriodico THEN
   IF BuscaLista1(DIA,MES,ANIO,HORA,MIN,SEC,Aux1) THEN
      AuxMen := Aux1^.MenPtr^.Next;
      WHILE (AuxMen <> Aux1^.MenPtr) AND
            (CompareStr(AuxMen^.Men,"cabeza3")<>0)DO
          IF AuxMen^.Men[0] = '1' THEN
            FOR e:=1 TO Length(AuxMen^.Men)-1 DO
                WritePuerto(AuxMen^.Men[e]);
            END (* for *);
          ELSIF
             AuxMen^.Men[0] = '5' THEN
              SetDeviceStatus(6,FALSE);
              DEPOSITA(AuxMen^.Men);
              SetDeviceStatus(6,TRUE);
          ELSE
            FOR e:=4 TO 79 DO
```

```
                    screen[24,e].char := ' ';
                  END (* for *);
                  FOR e := 0 TO Length(AuxMen^.Men) DO
                    screen[24,e+4].char := AuxMen^.Men[e];
                  END (* for *);
                END (* if *);
              AuxMen := AuxMen^.Next;
            END (* while *);
          END;
        END (* if *);
        (* Checar si se cumplio el tiempo de alguna lista con mensajes
           periodicos, para esto, primero se decrementa el contador de
           la lista en uno, despues se compara, si es cero entonces se
           insertan los mensajes que contenga en el buffer circular de
           buffer1 y despues se restablece el valor original del con--
           tador, esto se hace para cada lista periodica.              *)
        IF PrimerPeriodico^.Next <> PrimerPeriodico THEN
          Aux := PrimerPeriodico^.Next;
          REPEAT
            Aux^.AuxTiempoPer := Aux^.AuxTiempoPer - 1.0;
            IF Aux^.AuxTiempoPer = 0.0 THEN
              Aux^.AuxTiempoPer := Aux^.TiempoPer;
              AuxMen            := Aux^.MenPtr^.Next;
              WHILE AuxMen <> Aux^.MenPtr DO
                IF AuxMen^.Men[0] = '1' THEN
                  FOR e:=1 TO Length(AuxMen^.Men)-1 DO
                      WritePuerto(AuxMen^.Men[e]);
                  END (* for *)
                ELSIF
                   AuxMen^.Men[0] = '5' THEN
                     SetDeviceStatus(6,FALSE);
                     DEPOSITA(AuxMen^.Men);
                     SetDeviceStatus(6,TRUE);
                ELSE
                  FOR e:=4 TO 79 DO
                    screen[24,e].char := ' ';
                  END (* for *);
                  FOR e := 0 TO Length(AuxMen^.Men) DO
                    screen[24,e+4].char := AuxMen^.Men[e];
                  END (* for *);
                END (* if *);
                AuxMen := AuxMen^.Next;
              END (* while *);
            END (* if *);
            Aux := Aux^.Next;
          UNTIL Aux = PrimerPeriodico ;
        END(* if *) ;
      END; (* if tick = segundo *)
    CODE(0FAH);
    OUTBYTE(20H,20H);
    CODE(07H);   (* POP ES *)
    CODE(1FH);   (* POP DS *)
    CODE(5FH);   (* POP DI *)
    CODE(5EH);   (* POP SI *)
    CODE(5BH);   (* POP BX *)
```

```
    CODE(5AH);   (* POP DX *)
    CODE(59H);   (* POP CX *)
    CODE(58H);   (* POP AX *)

    CODE(89H, 0ECH);    (* MOV SP, BP *)
    CODE(5DH);          (* POP BP     *)
    CODE(0CFH);         (* IRET       *)
  END DeviceDriver;
  (*$S=*)
  (*$T=*)
  (*$R=*)
 (* regresamos la opcion del stack a su estado previo *)

  PROCEDURE ComienzaTimer();
  BEGIN
      SaveInterruptVector(InterruptVectorNumber,OldInterruptVector);
      (* guarda el estado del vector de interrupcion usado por el
         teclado.
      *)
      GetDeviceStatus(device,OldDeviceStatus);
      (* salva el estado del dispositivo
         (interrupciones habilitadas o desabilitadas )
      *)
      RestoreInterruptVector(InterruptVectorNumber,ADDRESS(DeviceDriver));
      (* cuelgo rutina que atienden interrupciones del reloj *)

      SetDeviceStatus(device,TRUE);
      (* habilita interrupciones del teclado *)
  END ComienzaTimer;

  PROCEDURE InicializaTiempo() ;
  BEGIN

     DameHoraSistema(HORA,MIN,SEC);
     DameFechaSistema(ANIO,MES,DIA);
     Delete(t,0,20);
  END InicializaTiempo;

  PROCEDURE ParaTimer;
  BEGIN
      SetDeviceStatus(device,OldDeviceStatus);
      (* Restablece el estado del dispositivo
         Habilitado/ Desabilitado.
      *)
      RestoreInterruptVector(InterruptVectorNumber,OldInterruptVector);
       (* restablece el valor original del vector
         de interrupcion usado por el dispositivo
         (teclado)                                  *)
  END ParaTimer;

BEGIN
  out := 0;
  Tick:= 0;
  TermProcedure(ParaTimer);
  activ:=FALSE;
```

```
  InicializaTiempo;
  VerifMensPuerto[0]:='1';
END TIMER.
```

```
DEFINITION MODULE Puerto1;
(*
Title    : PUERTO1
LastEdit: 23 JUNIO DE 1987
Author   : Felipe Verdalet Guzman
System   : LOGITECH MODULA-2/86
*)


EXPORT QUALIFIED StartReading,StopReading,out,in,
                 buffer,BufferSize;

CONST
       BufferSize = 4000; (* define el tamano del buffer donde deposito
                             los caracteres que envia el PC *)
VAR
    buffer          : ARRAY[0..BufferSize - 1] OF CHAR;
    in,out          :INTEGER; (* apuntadores al buffer *)

  PROCEDURE StartReading;
  (* instala una rutina de atencion a la interrupcion del
     puerto, asi mismo cambia la prioridad de las
     atencion a las interrupciones haciendo la del nivel
     3 la mas baja y la del 4 la mas alta. La interrupcion de
     nivel 4 es la del puerto primario.*)

  (* Regresa  los valores alterados a su valor original *)
  PROCEDURE StopReading ;

END Puerto1.
```

```
IMPLEMENTATION MODULE Puerto1;

(*
TitIe    : Puerto1
Comment : El presente modulo instrumenta un manejador de dispositivos
          en este caso se trata del Puerto serie. la rutina se amarra
          al vector que atiende las interrupciones del puerto serie
          primario OCH.

Nombre              | puerto 1 | puerto 2 | comentario
--------------------------------------------------------
LowBaudRegister     |   3F8H   |   2F8H   | Con el bit 7 en 1 del LineCntlReg
HighBaudRegister    |   3F9H   |   2F9H   | Con el bit 7 en 1 del LineCntlReg
ReceiverRegister    |   3F8H   |   2F8H   | Con el bit 7 en 0 del LineCntlReg
TransmitRegister    |   3F8H   |   2F8H   | Con el bit 7 en 0 del LineCntlReg
IntEnableRegister   |   3F9H   |   2F9H   | Con el bit 7 en 0 del LineCntlReg
IntIdentRegister    |   3FAH   |   2FAH   |
LineCntIRegister    |   3FBH   |   2FBH   |
ModemCntlRegister   |   3FCH   |   2FCH   |
LineStatRegister    |   3FDH   |   2FDH   |
ModemStatRegister   |   3FEH   |   2FEH   |
--------------------------------------------------------
LastEdit: 11 de Junio de 1987
Author  : Felipe Verdalet Guzman.
System  : LOGITECH MODULA-2/86
*)
 FROM SYSTEM IMPORT
    PROCESS, NEWPROCESS, TRANSFER, IOTRANSFER, ADR, SIZE,
    BYTE, ADDRESS, INBYTE, OUTBYTE, CODE;


 FROM System IMPORT
    TermProcedure;

 FROM Devices IMPORT
   GetDeviceStatus, SetDeviceStatus, SaveInterruptVector,
   RestoreInterruptVector;


 CONST                                  (*************************************
       device = 4;                      (* numero de bit en la mascara del
                                        (* controlador de interrupciones.

       InterruptVectorNumber = 0CH;     (* Numero del tipo de interrupcion
                                        (* Tipo de interrupcion generada por
                                        (* el RS-232

 VAR
   portInUse : CARDINAL;
   LowBaudReg, HighBaudReg, ReceiverReg, TransmitReg, IntEnablReg,
   IntIdentReg, LineCntlReg, ModmCntlReg, LineStatReg, ModmStatReg :
     CARDINAL;

   OldInterruptVector:ADDRESS;          (* guarda un vector de interrupcion
```

```
  OldDeviceStatus:BOOLEAN;          (* guarda el estado del dispositivo.
                                    (*  (interrupciones habilitadas o no)

PROCEDURE InitPort(port, baud, dataBits, stopBits, parity : CARDINAL);
  VAR
    param : BITSET;
    low, high : CARDINAL;
  BEGIN
    IF (port<>2) THEN
      port := 1;
    END;
    IF (port=2) THEN
      LowBaudReg := 2F8H;
    ELSE
      LowBaudReg := 3F8H;
    END;
    portInUse := port;

    ReceiverReg := LowBaudReg;
    TransmitReg := LowBaudReg;
    HighBaudReg := LowBaudReg+1;
    IntEnablReg := LowBaudReg+1;
    IntIdentReg := LowBaudReg+2;
    LineCntlReg := LowBaudReg+3;
    ModmCntlReg := LowBaudReg+4;
    LineStatReg := LowBaudReg+5;
    ModmStatReg := LowBaudReg+6;
    high := 0;
    CASE baud OF
      300 :
      low := 80H;
      high := 1H;
    | 600 :
      low := 0C0H;
    | 1200 :
      low := 60H;
    | 2400 :
      low := 30H;
    | 4800 :
      low := 18H;
    | 9600 :
      low := 0CH;
    ELSE
      low := 60H;
    END;
    OUTBYTE(LineCntlReg, 80H);
    OUTBYTE(LowBaudReg, low);
    OUTBYTE(HighBaudReg, high);
    param := {1,0};
    IF (dataBits=7) THEN
      EXCL(param, 0);
    END;
    IF (stopBits=2) THEN
      INCL(param, 2);
    END;
```

```
    IF (parity<>0) THEN
      INCL(param, 3);
      IF (NOT ODD(parity)) THEN
        INCL(param, 4);
      END;
    END;
    OUTBYTE(LineCntlReg, CARDINAL(param));
  END InitPort;


PROCEDURE DeviceDriver;
BEGIN
   (* prologo al procedimiento generado por el compilador:
        PUSH BP
        SUB SP, espacio variable
  *)
  (* salvamos todos los registros *)
  CODE(50H);   (* PUSH AX *)
  CODE(51H);   (* PUSH CX *)
  CODE(52H);   (* PUSH DX *)
  CODE(53H);   (* PUSH BX *)
  CODE(56H);   (* PUSH SI *)
  CODE(57H);   (* PUSH DI *)
  CODE(1EH);   (* PUSH DS *)
  CODE(06H);   (* PUSH ES *)
     INBYTE(3F8H,Dato);        (* lee el dato del registro Rx *)
     buffer[in] := Dato;
     in := (in + 1) MOD BufferSize;
  OUTBYTE(20H,20H);
  CODE(07H);   (* POP ES *)
  CODE(1FH);   (* POP DS *)
  CODE(5FH);   (* POP DI *)
  CODE(5EH);   (* POP SI *)
  CODE(5BH);   (* POP BX *)
  CODE(5AH);   (* POP DX *)
  CODE(59H);   (* POP CX *)
  CODE(58H);   (* POP AX *)

  CODE(89H, 0ECH);    (* MOV SP, BP *)
  CODE(5DH);          (* POP BP     *)
  CODE(0CFH);         (* IRET       *)
END DeviceDriver;
 (*$S=*)
 (*$T=*)
 (*$R=*)
(* regresamos la opcion del stack a su estado previo *)

 PROCEDURE StartReading;

   VAR
      ch:CHAR;

 BEGIN
     SaveInterruptVector(InterruptVectorNumber,OldInterruptVector);
     (* guarda el estado del vector de interrupcion usado por el
```

```
    puerto primario                                                    *)

  GetDeviceStatus(device,OldDeviceStatus);
  (* salva el estado del dispositivo
     (interrupciones habilitadas o desabilitadas )*)

  RestoreInterruptVector(InterruptVectorNumber,ADDRESS(DeviceDriver));
  (* asigna la direccion de la rutina que dara atencion a la interrupcio
     del puerto *)

  SetDeviceStatus(device,TRUE);     (* habilita interrupciones *)
  SetDeviceStatus(3,TRUE);          (* habilita interrupciones *)
  INBYTE(3F8H,ch);                  (* limpia el registro receptor *)
  CODE(0FAH);                       (* DESABILITA INTERRUPCIONES *)
  OUTBYTE(3FCH,08H);          (*habilito out2, REGISTRO DE CONTROL DE MODE

  OUTBYTE(3F9H,01H);          (* permite interrupciones del rs-232*)
  OUTBYTE(21H,0A4H);          (* ENMASCARO IR04 EN EL 8259A        *)
  (* HACEMOS UNA ROTACION DE PRIORIDADES HACIENDO LA PRIORIDAD
     DE NIVEL TRES LA MAS BAJA Y LA DE NIVEL 4 (* PUERTO *) MAS
     ALTA.
     OCW2 = R SL EOI 0 0 L2 L1 L0
     R,SL,EOI.- Estos tres bits controlan los modos
                Rotate y End Of Interrup  y combinacion de
                los dos.
     L2,L1,L0 .- Estos bits determinan el nivel de interrupcion
                 sobre el que se actuara cuando el SL esta activo.
     El comando para poner la prioridad es enviado a traves de OCW2
     donde : R = 1, SL = 1 ; L0-L2 es el nivel de prioridad binario
     de la unidad que se pondra con la mas baja prioridad, con lo cual
     todas las prioridades quedaran rotadas.
     por tanto la palabra de control para que el puerto primario quede
     con la prioridad mas alta es 'siguiendo el orden de  bits en OCW2
     1 1 0 0 0 0 1 1 = C3H.
  *)
  OUTBYTE(20H,0C3H);

  CODE(0FBH);                       (* HABILITA INTERRUPCIONES  *)

END StartReading;

PROCEDURE StopReading ;
BEGIN
  CODE(0FAH);                 (* deshabilitamos interrupciones *)

  OUTBYTE(20H,0C7H);          (* restablecemos mayor prioridad al timer *)
  OUTBYTE( 21H,0A4H);         (* restablece programacion del 8259 *)
  OUTBYTE(IntEnablReg,0H);
  OUTBYTE(ModmCntlReg,0H);

  SetDeviceStatus(device,OldDeviceStatus);
  (* restablece el estado original del puerto primario
     interrupciones habilitadas/deshabilitadas *)

  RestoreInterruptVector(InterruptVectorNumber,OldInterruptVector);
  (* restablece la rutina original que da servicio a las interrupciones
```

```
          del puerto primario *)

        CODE(0FBH);                (* habilita interrupciones *)

    END StopReading;

    PROCEDURE InitDevice ;
    BEGIN
      InitPort(1,9600,8,2,0);
    END InitDevice;
BEGIN
  TermProcedure(StopReading);
  (* instala 'StopDevice' como una rutina de terminacion
     en orden para el apropiado 'fin' del manejador de unidades

     cuando el programa que usa el manejador termina          *)
  in:=0;out:=0;
  InitDevice;
END Puerto1.
```

```
DEFINITION MODULE TECLADO;
   (* éste módulo exporta los procedimientos necesarios para
      el uso del teclado.
      Felipe Verdalet Guzmán
   *)
   EXPORT QUALIFIED  StartDevice, StopDevice;
   PROCEDURE StartDevice();
   (* Este procedimiento activa el manejador de interrupciones
      del teclado.

      Una vez activado el manejador realiza la siguiente tarea:

      cada 32 caracteres o cada return deposita el
      mensaje formado en un buffer que es administrado por el
      modulo monitor buffer1 de acuerdo con el algoritmo del
      barbero( Dijkstra ). Si se teclea <Esc> se borra el mensaje
      y no se deposita nada. Despues de depositar el mensaje, es
      borrado .
    *)

    PROCEDURE StopDevice();
    (* Este procedimiento desactiva el manejador de interrupciones
    *)

END TECLADO.
```

```
IMPLEMENTATION MODULE TECLADO;
(*
Title    : teclado
Comment : El presente modulo instrumenta un manejador interrupciones
          del teclado. Cada vez que ocurre una interrupcion se llama
          a la rutina KB_INT del BIOS para que procese el Scan Code
          y me deje el codigo ascii en el KB_BUFFER, despues tomo el
          codigo del buffer ayudandome de los apuntadores que maneja
          el BIOS Buffer_Head y Buffer_Tail. Solo Meto a buffer de
          mensajes los codigos >31. Un <ESC> anula el mensaje que se
          estaba construyendo, un <CR> deposita en el buffer general
          el mensaje construido, si este existe. todos los caracteres
          de control se reconocen, por tanto es posible establecer
          acciones para cada uno.

          ------------------------------------------------------
LastEdit: 23 DE JUNIO DE 1987
Author   : Felipe Verdalet Guzman.
System   : LOGITECH MODULA-2/86
*)
 FROM Strings IMPORT  Delete;
 FROM buffer1 IMPORT  DEPOSITA,mensaje, TamaMens;
 FROM SYSTEM IMPORT
    PROCESS, NEWPROCESS, TRANSFER, IOTRANSFER, ADR, SIZE,
    BYTE,WORD, ADDRESS, INBYTE,OUTBYTE, CODE, SWI,ENABLE,DISABLE;

 FROM System IMPORT
    TermProcedure;

 FROM Devices IMPORT
   GetDeviceStatus, SetDeviceStatus, SaveInterruptVector,
   RestoreInterruptVector;

 FROM Screen IMPORT
  Write,WriteLn, WriteString,WriteHex,WriteCard, BackSpace;

 TYPE                                    (*************************************
      CARACTER =                         (* tipo auxiliar para contener el
        RECORD                           (* caracter leido de el KB_BUFFER
        CASE BOOLEAN OF                  (*************************************
          TRUE :
          X : CARDINAL;
        | FALSE :
          Y,Z:CHAR;
        END;
        END;
TYPE
    ScreenType =
           ARRAY[0..24] OF              (* filas *)
           ARRAY[0..79] OF              (* columnas *)
           RECORD char,attr:CHAR END;   (* contenido *)
 VAR
     screen[0B800H:0H]:ScreenType;      (* arreglo para accesar el video *)
     e:CARDINAL;
```

```
CONST                                   (*************************************
      device = 1;                       (* numero de bit en la mascara del
                                        (* controlador de interrupciones.

      InterruptVectorNumber = 09H;      (* Numero del tipo de interrupcion
                                        (* generada por el TECLADO
 VAR

     OldInterruptVector:ADDRESS;        (* guarda un vector de interrupcion

     OldInterruptVector60H:ADDRESS;     (* guarda un vector de interrupcion

     OldDeviceStatus:BOOLEAN;           (* guarda el estado del dispositivo
                                        (*  (interrupciones habilitadas o no)

     BUFFERHEAD[0040H:001AH] : CARDINAL;(* apuntador a la cabeza del KB_BUFF
     BUFFERTAIL[0040H:001CH] : CARDINAL;(* apuntador a la cola del KB_BUFF
     MEMCARAC:ADDRESS;                  (* apunta al caracter a leer de KB_B
     BufferInterno : mensaje;           (* forma el mensaje del operador
     Dato          : CHAR;              (* toma el caracter presionado
     DATO1         : WORD;              (* auxiliar
     Point         : CARDINAL;
     CARACTER1     : CARACTER;

  (*$S-*)
  (*$T-*)
  (*$R-*)
(* Quitamos la prueba del stack porque este es llamado
   despues de salvar los registros, de tal modo que
   se pueden destruir *)

PROCEDURE DeviceDriver;
BEGIN
   (* prologo al procedimiento generado por el compilador:
         PUSH BP
         SUB SP, espacio variable
  *)
  (* guardamos todos los registros *)
  CODE(50H);   (* PUSH AX *)
  CODE(51H);   (* PUSH CX *)
  CODE(52H);   (* PUSH DX *)
  CODE(53H);   (* PUSH BX *)
  CODE(56H);   (* PUSH SI *)
  CODE(57H);   (* PUSH DI *)
  CODE(1EH);   (* PUSH DS *)
  CODE(06H);   (* PUSH ES *)
     CODE(55H);                         (* llamado a interrupcion del teclado *
     SWI(60H);                          (* KB_INT salvando el BP.             *
     CODE(5DH);

     IF BUFFERHEAD <> BUFFERTAIL THEN      (* si hay caracter en KB_BUFFER *
       ENABLE;                                (* habilita interrupciones *
       MEMCARAC.OFFSET    := BUFFERHEAD;      (* obtiene la direccion    *
       DATO1              := MEMCARAC^;       (* donde esta el caracter  *
       CARACTER1.X         := CARDINAL(DATO1); (* la pasa a variables aux *
```

```
Dato                := CARACTER1.Y;          (* luego a Dato, e iguala  *
BUFFERHEAD          := BUFFERTAIL;           (* apuntadores de KB_BUFF. *

CASE ORD(Dato) OF
  00:
| 01:                                        (* Cntl^A                  *
     FOR e:=4 TO 79 DO                       (* borra zona de alarmas   *
       screen[23,e].char := ' ';
     END (* for *);

| 02:                                        (* Cntl^B                  *
     FOR e:=4 TO 79 DO                       (* borra zona de mensajes  *
       screen[24,e].char := ' ';
     END (* for *);
| 03:
| 04:
| 05:
| 06:
| 07:
| 08:    IF Point > 1 THEN BackSpace;DEC(Point); END;
| 09:
| 10:
| 11:
| 12:
| 13:
    IF Point <> 1 THEN
     Delete(BufferInterno,Point,TamaMens-Point);
     DEPOSITA(BufferInterno);   (* mete el mensaje en el buffer     *
     Point := 1;                (* inicializa apuntador de inser-   *
    END;

| 14:
| 15:
| 16:
| 17:
| 18:
| 19:
| 20:
| 21:
| 22:
| 23:
| 24:
| 25:
| 26:
| 27:
    WHILE Point > 1 DO          (* borra el mensaje de pantalla y   *
      BackSpace;                (* del buffer                       *
      DEC(Point);
    END; (* while *)
| 28:
| 29:
| 30:
| 31:
ELSE
 Write(Dato);
```

```
        BufferInterno[Point] := Dato;(* mete el caracter al buffer        *)
        Point:=(Point + 1) MOD TamaMens; (* incrementa apuntador de inser.

        IF Point = 0 THEN Point:=1; END;
       END (* case *);                    .!
     END (* if buffer-head <> buffer-tail *);
  DISABLE;
  OUTBYTE(20H,20H);              (* envio de EOI al 8259-A *)
  CODE(07H);   (* POP ES *)
  CODE(1FH);   (* POP DS *)
  CODE(5FH);   (* POP DI *)
  CODE(5EH);   (* POP SI *)
  CODE(5BH);   (* POP BX *)
  CODE(5AH);   (* POP DX *)
  CODE(59H);   (* POP CX *)
  CODE(58H);   (* POP AX *)

  CODE(89H, 0ECH);    (* MOV SP, BP *)
  CODE(5DH);          (* POP BP     *)
  CODE(0CFH);         (* IRET       *)
END DeviceDriver;
 (*$S=*)
 (*$T=*)
 (*$R=*)
(* regresamos la opcion del stack a su estado previo *)

 PROCEDURE StartDevice;
 BEGIN

     SaveInterruptVector(InterruptVectorNumber,OldInterruptVector);
     (* guarda el estado del vector de interrupcion usado por el
        teclado.
     *)
     SaveInterruptVector(60H, OldInterruptVector60H);
     (* El tipo de interrupcion 60H es asignado por bios para
        uso general, de todas maneras se salva el valor que
        tiene, por si las moscas.
     *)
     RestoreInterruptVector(60H, OldInterruptVector);
     (* asigna al tipo de interrupcion 60H (Asignado para uso
        comun) la direccion de la rutina que da servicio al
        teclado KB_INT.
     *)

     RestoreInterruptVector(InterruptVectorNumber,ADDRESS(DeviceDriver));
     (* con esta instruccion cuelgo la rutina DeviceDriver al vector de
        interrupcion donde se encuentra originalmente KB_INT *)


     SetDeviceStatus(device,TRUE);
     (* habilita interrupciones del teclado *)

END StartDevice;

PROCEDURE StopDevice ;
BEGIN
```

```
      SetDeviceStatus(device,OldDeviceStatus);
      (* Restablece el estado del dispositivo
         Habilitado/ Desabilitado.
      *)
      RestoreInterruptVector(InterruptVectorNumber,OldInterruptVector);
       (* restablece el valor original del vector
         de interrupcion usado por el dispositivo
         (teclado)                                  *)

      RestoreInterruptVector(60H, OldInterruptVector60H);
        (* Devuelve el valor que tenia el tipo de interrupcion
           60H.
        *)

  END StopDevice;

BEGIN
  MEMCARAC.SEGMENT := 40H; (* apuntador a el buffer del teclado  *)
  BufferInterno[0]:='4';   (* identifica a un mensaje de teclado *)
  Point:= 1;               (* apuntador al buffer de mensajes    *)
  TermProcedure(StopDevice);
END TECLADO.
```

```
DEFINITION MODULE LISTAS;

(* exporta las variables, estructuras de datos y procedimientos
   para el manejo de listas doblemente ligadas con cabezera

   Felipe Verdalet Guzman
*)

  FROM buffer1 IMPORT mensaje;
  FROM SYSTEM IMPORT ADDRESS;
  EXPORT QUALIFIED  TiempoSeg,TiempoNoPerPtr,TiempoPtr,
                    MensajePtr,PrimerNoPeriodico,PrimerPeriodico,
                    InsertaAntes,BorraLista,BuscaLista1,BuscaLista2,
                    BuscaListaMayor1,BuscaListaMayor2,
                    CreaLista1,CreaLista2,CreaMensaje,InsertaMensaje1,
                    InsertaMensaje2,BorraMensaje1,BorraMensaje2;

   TYPE
       TiempoNoPerPtr = POINTER TO Tiempos1;(*apunta a tiempos no periodicos*
       TiempoPtr      = POINTER TO Tiempos; (*apunta a tiempos periodicos *)
       MensajePtr     = POINTER TO Mensajes;(* apunta a mensajes *)

       Tiempos = RECORD      (* periodicos *)
         Prev          : TiempoPtr;
         Next          : TiempoPtr;
         TiempoPer     : REAL;
         AuxTiempoPer  : REAL;
         MenPtr        : MensajePtr;
       END;   (* tiempos *)

       Tiempos1 = RECORD     (* no periodicos *)
         Prev       : TiempoNoPerPtr;
         Next       : TiempoNoPerPtr;
         Dia        : CARDINAL;
         Mes        : CARDINAL;
         Anio       : CARDINAL;
         Hora       : CARDINAL;
         Min        : CARDINAL;
         Sec        : CARDINAL;
         MenPtr     : MensajePtr;
       END; (* tiempos1 *)

       Mensajes = RECORD      (* mensaje *)
         Prev     : MensajePtr;
         Next     : MensajePtr;
         Men      : mensaje;
       END;   (* mensajes *)
 VAR
  NoperiodicoPtr,NoperiodicoAuxPtr,PrimerNoPeriodico:TiempoNoPerPtr;
  PeriodicoPtr,PeriodicoAuxPtr,PrimerPeriodico:TiempoPtr;


 PROCEDURE TiempoSeg(Hora,Min,Sec:CARDINAL):CARDINAL;

 (* convierte la hora en segundos *)
```

```
  PROCEDURE InsertaAntes(tipo: CARDINAL;ListaPtr, AntesDe:ADDRESS) ;
  (* inserta una lista apuntada por ListaPtr Antes de la lista
     apuntada por AntesDe. Si tipo = 1 es lista periodica, si tipo = 2
     es lista no periodica *)

  PROCEDURE BorraLista(ListaPtr:ADDRESS; tipo : CARDINAL;VAR Cabeza:CHAR):BOOLE
N ;
  (* borra la lista apuntada por ListaPtr del tipo x,  en caso de
     ser  cabeza  de  lista regresa la variable Cabeza  co  valor
     "s" si no la pudo borrar la funcion regresa un valor falso *)

  PROCEDURE BuscaLista1(dia,mes,anio,Hora1, Min1, Sec1: CARDINAL;
                        VAR Lista:ADDRESS):BOOLEAN ;
  (*  busca  la lista correspondiente a la fecha  y  hora  dados,
regresa un apuntador a esta. Si no la encuentra devuelve un valor
falso *)

  PROCEDURE BuscaLista2(Tiempo : REAL; VAR Lista:ADDRESS): BOOLEAN ;
   (*  busca  una lista con el tiempo "Tiempo",  si la  encuentra
regresa  un  apuntador a esta.  Si no la  encuentra,  la  funcion
regresa un valor falso *)

  PROCEDURE BuscaListaMayor1(dia,mes,anio,
                             Hora1, Min1, Sec1: CARDINAL; VAR Lista:ADDRESS) ;

  (* busca una lista con fecha mayor que la dada, si la encuentra
regresa  un apuntador a esta en "Lista".  *)

  PROCEDURE BuscaListaMayor2(Tiempo : REAL; VAR Lista:ADDRESS) ;

(* busca una lista periodica que tenga un tiempo mayor al dado en
"Tiempo".  Si  la  encuentra  regresa un apuntador a esta  en  la
variable "Lista". *)

  PROCEDURE CreaLista1(dia1,mes1,anio1,
                       Hora1,Min1,Sec1:CARDINAL; VAR ListaPtr:ADDRESS) ;

(* crea una lista no periodica, regresa un apuntador a esta lista
en "ListaPtr". *)

  PROCEDURE CreaLista2(Sec1:REAL; VAR ListaPtr : ADDRESS) ;

(*  crea una lista periodica,  regresa un apuntador a esta en  la
variable "ListaPtr" *)

  PROCEDURE CreaMensaje(mens:mensaje ; VAR Nuevomensaje:ADDRESS) ;
(*  crea  un  registro "mensaje",  regresa un  apuntador  a  este
registro en la variable "Nuevomensaje" *)

  PROCEDURE InsertaMensaje1(mensajeDado:mensaje;dia,mes,anio,
                            hora,min,seg:CARDINAL);

  (*  inserta un mensaje en una lista no periodica con la hora  y
fecha  dada,si la lista no existe,  la crea y despues inserta  el
mensaje *)
```

```
  PROCEDURE InsertaMensaje2(mensajeDado:mensaje;tiempo:REAL) ;
(*  inserta un mensaje en una lista periodica,  si no  existe  la
lista periodica, la crea *)

  PROCEDURE BorraMensaje1(mensaje1:mensaje;dia,mes,anio,
                                hora,min,seg:CARDINAL):BOOLEAN;
 (* borra un mensaje de una lista no periodica *)

  PROCEDURE BorraMensaje2(mensaje1:mensaje;tiempo:REAL):BOOLEAN ;
  (* borra un mensaje de una lista periodica *)

END LISTAS.
```

```
IMPLEMENTATION MODULE LISTAS;
(*
Title    : LISTAS
Comment : Este modulo tiene como fin el implementar
          el manejo de listas doblemente ligadas. las
          cuales a su vez contienen listas doblemente
          ligadas.
LastEdit: 02 ABRIL 1987
Author   : Felipe Verdalet Guzman
System   : LOGITECH MODULA-2/86
*)
FROM buffer1 IMPORT mensaje;
FROM utilerias IMPORT ConvFechaADias,ConvHoraAHoras,SumaFechaHora;
FROM NumberConversion IMPORT StringToCard,CardToString;
FROM Storage IMPORT ALLOCATE,DEALLOCATE;
FROM SYSTEM IMPORT ADDRESS;
FROM Strings IMPORT CompareStr;
PROCEDURE TiempoSeg(Hora,Min,Sec:CARDINAL):CARDINAL;
  (* Convierte una hora de tipo HH:MM:SS en Segundos *)
BEGIN
  RETURN(Hora*3600 + Min * 60 + Sec);
END TiempoSeg;


  PROCEDURE InsertaAntes(tipo : CARDINAL;
                         ListaPtr:ADDRESS;
                         AntesDe:ADDRESS) ;
  (* Este procedimiento se encarga de insertar una lista antes de
     la lista AntesDe la cual se supone fue encontrada con un
     procedimiento de busquedas.
  *)
   VAR
      ListaTipo1,ListaTipo11:TiempoNoPerPtr;
      ListaTipo2,ListaTipo22:TiempoPtr;
      EsMensaje, EsMensaje1 :MensajePtr;

   BEGIN

     CASE tipo OF
       1:
          ListaTipo1:= ListaPtr;
          ListaTipo11:=AntesDe;
          ListaTipo1^.Next         := ListaTipo11;
          ListaTipo1^.Prev         := ListaTipo11^.Prev;
          ListaTipo11^.Prev^.Next := ListaTipo1;
          ListaTipo11^.Prev        := ListaTipo1;
      |2:
          ListaTipo2:= ListaPtr;
          ListaTipo22:=AntesDe;
          ListaTipo2^.Next         := ListaTipo22;
          ListaTipo2^.Prev         := ListaTipo22^.Prev;
          ListaTipo22^.Prev^.Next := ListaTipo2;
          ListaTipo22^.Prev        := ListaTipo2;
      |3:
          EsMensaje:= ListaPtr;
```

```
        EsMensaje1:=AntesDe;
        EsMensaje^.Next          := EsMensaje1;
        EsMensaje^.Prev          := EsMensaje1^.Prev;
        EsMensaje1^.Prev^.Next   := EsMensaje;
        EsMensaje1^.Prev         := EsMensaje;
    END;
 END InsertaAntes;

PROCEDURE BorraLista( ListaPtr:ADDRESS; tipo : CARDINAL;VAR Cabeza:CHAR):BO
L ;
  (* Este procedimiento borra un elemento de una lista, o una
     lista de una lista de listas dado unicamente la direccion
     del elemento a borrar *)
  VAR
       ListaTipo1:TiempoNoPerPtr;
       ListaTipo2:TiempoPtr;
       EsMensaje :MensajePtr;
       Aux:ADDRESS;
BEGIN
   Cabeza := 'N';
   IF (ListaPtr = PrimerPeriodico) OR (ListaPtr = PrimerNoPeriodico) THEN
     RETURN(FALSE);
   END;
   CASE tipo OF

     1:
       ListaTipo1:=ListaPtr;
       ListaTipo1^.Prev^.Next := ListaTipo1^.Next;
       ListaTipo1^.Next^.Prev := ListaTipo1^.Prev;
       WHILE (ListaTipo1^.MenPtr <> NIL) AND
             (CompareStr(ListaTipo1^.MenPtr^.Men,"cabeza3")<>0) DO
         Aux := ListaTipo1^.MenPtr^.Next;
         DISPOSE(ListaTipo1^.MenPtr);
         ListaTipo1^.MenPtr := Aux;
       END (* while *);
       IF CompareStr(ListaTipo1^.MenPtr^.Men,"cabeza3") = 0 THEN
          DISPOSE(ListaTipo1^.MenPtr);

       END (* if *);

       DISPOSE(ListaTipo1);

       RETURN(TRUE);
    2:
       ListaTipo2:=ListaPtr;
       ListaTipo2^.Prev^.Next := ListaTipo2^.Next;
       ListaTipo2^.Next^.Prev := ListaTipo2^.Prev;
       WHILE (ListaTipo2^.MenPtr <> NIL) AND
             (CompareStr(ListaTipo2^.MenPtr^.Men,"Cabeza3")<>0) DO
         Aux := ListaTipo2^.MenPtr^.Next;
         DISPOSE(ListaTipo2^.MenPtr);
         ListaTipo2^.MenPtr := Aux;
       END (* while *);
       IF CompareStr(ListaTipo2^.MenPtr^.Men,"Cabeza3") = 0 THEN
           DISPOSE(ListaTipo2^.MenPtr);
       END (* if *);
```

```
      DISPOSE(ListaTipo2);
      RETURN(TRUE);
    |3:
      EsMensaje:=ListaPtr;
      EsMensaje^.Prev^.Next := EsMensaje^.Next;
      EsMensaje^.Next^.Prev := EsMensaje^.Prev;
      IF EsMensaje^.Next = EsMensaje^.Prev THEN Cabeza := 'S'; END;
      DISPOSE(EsMensaje);
      RETURN(TRUE);
  END;  (* case *)
END BorraLista;

PROCEDURE BuscaLista1(dia,mes,anio,
                      Hora1, Min1, Sec1: CARDINAL;
                      VAR Lista:ADDRESS):BOOLEAN ;
   (* Este Procedimiento se encarga de buscar una lista dada en la lista
      de listas de los mensajes no periodicos y regresa un apuntador a
      esa lista en la variable Lista. En caso de no encontrarla regresa
      de la funcion con un valor falso *)
   VAR
       Aux:TiempoNoPerPtr;
       horas,mins,secs:ARRAY [0..1] OF CHAR;
       Horas,Sumahoras:REAL;
       sumadias,Sumadias1:CARDINAL;
       Sumadiashoras,Sumadiashoras1:REAL;
       HoraString,MinString,SecString:ARRAY[0..1] OF CHAR;
BEGIN
  Aux := PrimerNoPeriodico^.Next;
  IF Aux = PrimerNoPeriodico THEN
    RETURN(FALSE);
  END;
  (* convertimos la hora en string para despues
     convertirla a real. no he visto como convertir de
     cardinal a real directamente *)
  CardToString(Hora1,horas,2);
  CardToString(Min1,mins,2);
  CardToString(Sec1,secs,2);

  ConvHoraAHoras(horas,mins,secs,Horas);
  ConvFechaADias(dia,mes,anio,sumadias);
  SumaFechaHora(sumadias,Horas,Sumadiashoras);

  CardToString(Aux^.Hora,HoraString,2);
  CardToString(Aux^.Min,MinString,2);
  CardToString(Aux^.Sec,SecString,2);

  ConvHoraAHoras(HoraString,MinString,SecString,Sumahoras);
  ConvFechaADias(Aux^.Dia,Aux^.Mes,Aux^.Anio,Sumadias1);
  SumaFechaHora(Sumadias1,Sumahoras,Sumadiashoras1);

  WHILE (Sumadiashoras <> Sumadiashoras1) AND
        (Aux <> PrimerNoPeriodico) DO
         Aux := Aux^.Next;
         IF Aux <> PrimerNoPeriodico THEN
```

```
            CardToString(Aux^.Hora,HoraString,2);
            CardToString(Aux^.Min,MinString,2);
            CardToString(Aux^.Sec,SecString,2);

            ConvHoraAHoras(HoraString,MinString,SecString,Sumahoras);
            ConvFechaADias(Aux^.Dia,Aux^.Mes,Aux^.Anio,Sumadias1);
            SumaFechaHora(Sumadias1,Sumahoras,Sumadiashoras1);
          END; (* if *)
    END (* while *);
      IF Aux=PrimerNoPeriodico THEN RETURN(FALSE) END;
      Lista:=Aux;
      RETURN(TRUE);
  END BuscaLista1;

PROCEDURE BuscaLista2(Tiempo : REAL; VAR Lista:ADDRESS): BOOLEAN ;
    (* Este procedimiento se encarga de encontrar una lista dada en
       la lista de listas de mensajes periodicos, en caso de encontrarla
       regresa su apuntador en la variable Lista, En caso contrario
       la funcion regresa con un valor de falso *)
 VAR
      Aux:TiempoPtr;

  BEGIN

    Aux := PrimerPeriodico^.Next;
    IF Aux = PrimerPeriodico THEN
      RETURN(FALSE);
    END;
    WHILE (Tiempo <> Aux^.TiempoPer)      AND
          (Aux <> PrimerPeriodico) DO
          Aux := Aux^.Next;
    END (* while *);
    IF Aux = PrimerPeriodico THEN
     RETURN(FALSE);
    ELSE
      Lista:=Aux;
      RETURN(TRUE);
    END (* if *);
  END BuscaLista2;

  PROCEDURE BuscaListaMayor1(dia,mes,anio,
                             Hora1, Min1, Sec1: CARDINAL; VAR Lista:ADDRESS)
     (* Este Procedimiento se encarga de buscar una lista dada en la lista
        de listas de los mensajes no periodicos Cuyo tiempo sea mayor que el
        dado como parametro si la encuentra regresa un apuntador a
        esa lista en la variable Lista. Si no lo encuentra, el apuntador
        sera a la cabeza de lista *)
     VAR
         Aux :TiempoNoPerPtr;
         horas,mins,secs:ARRAY [0..1] OF CHAR;
         Horas,Sumahoras:REAL;
         sumadias,Sumadias1:CARDINAL;
         Sumadiashoras,Sumadiashoras1:REAL;
         HoraString,MinString,SecString:ARRAY[0..1] OF CHAR;
```

```
  BEGIN

    Aux := PrimerNoPeriodico^.Next;
    IF Aux = PrimerNoPeriodico THEN Lista := Aux;
    ELSE
      (* convertimos la hora en string para despues
         convertirla a real. no he visto como convertir de
         cardinal a real directamente *)
      CardToString(Hora1,horas,2);
      CardToString(Min1,mins,2);
      CardToString(Sec1,secs,2);
      ConvHoraAHoras(horas,mins,secs,Horas);
      ConvFechaADias(dia,mes,anio,sumadias);
      SumaFechaHora(sumadias,Horas,Sumadiashoras);

      CardToString(Aux^.Hora,HoraString,2);
      CardToString(Aux^.Min,MinString,2);
      CardToString(Aux^.Sec,SecString,2);
      ConvHoraAHoras(HoraString,MinString,SecString,Sumahoras);
      ConvFechaADias(Aux^.Dia,Aux^.Mes,Aux^.Anio,Sumadias1);
      SumaFechaHora(Sumadias1,Sumahoras,Sumadiashoras1);

      WHILE    (Sumadiashoras > Sumadiashoras1) AND
               (Aux <> PrimerNoPeriodico)                    DO
               Aux:= Aux^.Next;
               IF Aux <> PrimerNoPeriodico THEN
                 CardToString(Aux^.Hora,HoraString,2);
                 CardToString(Aux^.Min,MinString,2);
                 CardToString(Aux^.Sec,SecString,2);
                 ConvHoraAHoras(HoraString,MinString,SecString,Sumahoras);
                 ConvFechaADias(Aux^.Dia,Aux^.Mes,Aux^.Anio,Sumadias1);
                 SumaFechaHora(Sumadias1,Sumahoras,Sumadiashoras1);
               END; (* if *)
      END (* while *);
          Lista:=Aux;
    END; (* IF *)
  END BuscaListaMayor1;

PROCEDURE BuscaListaMayor2(Tiempo : REAL; VAR Lista:ADDRESS) :
     (* Este procedimiento se encarga de encontrar una lista dada en
        la lista de listas de mensajes periodicos, en caso de encontrarla
        regresa su apuntador en la variable Lista, si no lo encuentra
        el apuntador sera a la cabeza de lista *)
      VAR
          Aux :TiempoPtr;

  BEGIN
    Aux := PrimerPeriodico^.Next;
    WHILE ((Aux <> PrimerPeriodico) AND (Aux^.TiempoPer < Tiempo)) DO
          Aux := Aux^.Next;
    END (* while *);
    Lista:=Aux;
  END BuscaListaMayor2;
```

```
PROCEDURE CreaLista1( dia1,mes1,anio1,
                      Hora1,Min1,Sec1:CARDINAL; VAR ListaPtr:ADDRESS) ;
  VAR
     MensajeAuxPtr : MensajePtr;
BEGIN
    NEW(NoperiodicoPtr);
    WITH  NoperiodicoPtr^ DO
       NEW(MenPtr);
       Next        := NoperiodicoPtr;
       Prev        := NoperiodicoPtr;
       Dia         := dia1;
       Mes         := mes1;
       Anio        := anio1;
       Hora        := Hora1;
       Min         := Min1;
       Sec         := Sec1;
       MenPtr^.Next := MenPtr;
       MenPtr^.Prev := MenPtr;
       MenPtr^.Men  := "cabeza3"
    END (* With *);
    ListaPtr := NoperiodicoPtr;
END CreaLista1;

PROCEDURE CreaLista2(Sec1:REAL; VAR ListaPtr : ADDRESS) ;
  VAR
    MensajeAuxPtr : MensajePtr;
BEGIN
    NEW(PeriodicoPtr);
    WITH PeriodicoPtr^ DO
      NEW(MenPtr);
      Next          := PeriodicoPtr;
      Prev          := PeriodicoPtr;
      TiempoPer     := Sec1;
      AuxTiempoPer  := Sec1;
      MenPtr^.Next  := MenPtr;
      MenPtr^.Prev  := MenPtr;
      MenPtr^.Men   := "Cabeza3"
    END (* With *);
    ListaPtr := PeriodicoPtr;
END CreaLista2 ;

PROCEDURE CreaMensaje(mens:mensaje; VAR Nuevomensaje:ADDRESS) ;
   (* Este Procedimiento tiene como entrada el mensaje que se quiere
      Crear y  como salida un apuntador al mensaje creado          *)
   VAR
       Aux :MensajePtr;
BEGIN
  NEW(Aux);
  WITH Aux^ DO
    Prev := NIL;
    Next := NIL;
    Men  := mens;
  END (* With *);
  Nuevomensaje := Aux;
END CreaMensaje;
```

```
PROCEDURE InsertaMensaje1(mensajeDado:mensaje;dia,mes,anio,
                          hora,min,seg:CARDINAL);
  (* este mensaje se encarga de insertar un mensaje en una
     lista de tipo no periodica, si la lista existe, se anade
     el mensaje a esta, si no existe, entonces se crea una lista
     y se inserta el mensaje en esta, despues se inserta la lista
     en la lista de listas no periodicas *)
  VAR
      MensPoint: MensajePtr;
      Lista1Ptr: TiempoNoPerPtr;
      ListaMay : TiempoNoPerPtr;
BEGIN
  CreaMensaje(mensajeDado,MensPoint);       (* creo el nodo mensaje       *)
  IF BuscaLista1(dia,mes,anio,hora,min,seg,Lista1Ptr) THEN
     WITH Lista1Ptr^ DO                     (* si existe la lista con     *)
       InsertaAntes(3,MensPoint,MenPtr);    (* el tiempo dado se inserta *)
     END;                                   (* el mensaje en ella.        *)
  ELSE                                      (* si no existe la lista      *)
    CreaLista1(dia,mes,anio,hora,min,seg,Lista1Ptr);
    WITH Lista1Ptr^ DO                      (* creo la lista e inserto el*)
      InsertaAntes(3,MensPoint,MenPtr);     (* mensaje en esta            *)
    END;                                    (* despues busco el lugar que*)
    BuscaListaMayor1(dia,mes,anio,hora,min,seg,ListaMay);
    InsertaAntes(1,Lista1Ptr,ListaMay);     (* le corresponde a esta lista*)
  END;                                      (* en la lista de listas y la *)
END InsertaMensaje1;                        (* inserto.                   *)

PROCEDURE InsertaMensaje2(mensajeDado:mensaje;tiempo:REAL) ;
  VAR
      MensPoint: MensajePtr;
      Lista1Ptr: TiempoPtr;
      ListaMay : TiempoPtr;
BEGIN
  CreaMensaje(mensajeDado,MensPoint);       (* creo el nodo mensaje       *)
  IF BuscaLista2(tiempo,Lista1Ptr) THEN
     WITH Lista1Ptr^ DO                     (* si existe la lista con     *)
       InsertaAntes(3,MensPoint,MenPtr);    (* el tiempo dado se inserta *)
     END;                                   (* el mensaje en ella.        *)
  ELSE                                      (* si no existe la lista      *)
    CreaLista2(tiempo,Lista1Ptr);
    BuscaListaMayor2(tiempo,ListaMay);
    InsertaAntes(2,Lista1Ptr,ListaMay);     (* le corresponde a esta lista*)
    WITH Lista1Ptr^ DO                      (* creo la lista e inserto el*)
      InsertaAntes(3,MensPoint,MenPtr);     (* mensaje en esta            *)
    END;                                    (* despues busco el lugar que*)
  END; (* if *)                            (* en la lista de listas y la *)
END InsertaMensaje2;                        (* inserto                    *)

  PROCEDURE BorraMensaje1(mensaje1:mensaje;dia,mes,anio,
                          hora,min,seg:CARDINAL):BOOLEAN;
    VAR
        Lista1Ptr:TiempoNoPerPtr;
        MensPoint: MensajePtr;
        Cabeza   : CHAR;
```

```
    BEGIN
      IF BuscaLista1(dia,mes,anio,hora,min,seg,Lista1Ptr) THEN
        MensPoint := Lista1Ptr^.MenPtr^.Next;
        WHILE (CompareStr(MensPoint^.Men,mensaje1)<>0) AND
              (MensPoint <> Lista1Ptr^.MenPtr) DO

              MensPoint := MensPoint^.Next;

        END; (* while *)
        IF MensPoint <> Lista1Ptr^.MenPtr THEN
          IF BorraLista(MensPoint,3,Cabeza) THEN
            IF Cabeza = 'S' THEN
              IF BorraLista(Lista1Ptr,1,Cabeza) THEN (* *) END;(* borra la li
pues *)
            END; (* if cabeza *)                                    (* quedo sin m
ajes  *)
            RETURN(TRUE);
          ELSE
            RETURN(FALSE);
          END (* if *);
        ELSE
          RETURN(FALSE);
        END (* if *);
      ELSE
        RETURN(FALSE);
      END (* if *);
 END BorraMensaje1;

    PROCEDURE BorraMensaje2(mensaje1:mensaje;tiempo:REAL):BOOLEAN ;
      VAR
          Lista1Ptr:TiempoPtr;
          MensPoint: MensajePtr;
          Cabeza :CHAR;
    BEGIN
      IF BuscaLista2(tiempo,Lista1Ptr) THEN
        MensPoint := Lista1Ptr^.MenPtr^.Next;

        WHILE (CompareStr(MensPoint^.Men,mensaje1) <> 0) AND
              (MensPoint <> Lista1Ptr^.MenPtr) DO
          MensPoint := MensPoint^.Next;
        END; (* while *)

        IF MensPoint <> Lista1Ptr^.MenPtr THEN
          IF BorraLista(MensPoint,3,Cabeza) THEN
            IF Cabeza = 'S' THEN                          (* si ya no hay mensaj
*)
              IF BorraLista(Lista1Ptr,2,Cabeza) THEN (* *) END;
            END; (* If Cabeza *)                          (* borro la lista
*)
            RETURN(TRUE);
          ELSE
            RETURN(FALSE);
          END (* if *);
        ELSE
          RETURN(FALSE);
        END (* if *);
      ELSE
        RETURN(FALSE);
      END (* if *);
```

```
    END BorraMensaje2;
BEGIN
  NEW(PrimerPeriodico);
  WITH PrimerPeriodico^ DO
    Next          := PrimerPeriodico;
    Prev          := PrimerPeriodico;
    TiempoPer     := 0.0;
    AuxTiempoPer  := 0.0;
    NEW(MenPtr);
    MenPtr^.Next  := MenPtr;
    MenPtr^.Prev  := MenPtr;
    MenPtr^.Men   := "Cabeza2";
  END; (* WITH *)
  NEW(PrimerNoPeriodico);
  WITH PrimerNoPeriodico^ DO
    Next          := PrimerNoPeriodico;
    Prev          := PrimerNoPeriodico;
    Dia           := 0;
    Mes           := 0;
    Anio          := 0;
    Hora          := 0;
    Min           := 0;
    Sec           := 0;
    NEW(MenPtr);
    MenPtr^.Next  := MenPtr;
    MenPtr^.Prev  := MenPtr;
    MenPtr^.Men   := "CABEZA2";
  END; (* With *)
END LISTAS.
```

```
MODULE Monitor2[0];
(*
Title    : Principal
Comment  : En este modulo se crean los procesos : Maestro ;
InterpretaMenPC; VerificaRespPC; Menupri.| Tambien se activan los
manejadores de interrupcion: Teclado; Reloj; Timer.

LastEdit: 8 de Junio de 1987.
Author   : Felipe Verdalet Guzman.
System   : LOGITECH MODULA-2/86
*)
  FROM utilerias IMPORT ConvFechaADias,ConvHoraAHoras,SumaFechaHora;
  FROM buffer1 IMPORT  SACA,mensaje;
  FROM RealConversions IMPORT StringToReal, RealToString;
  FROM NumberConversion IMPORT StringToCard,CardToString;
  FROM Screen IMPORT   GotoXY, Scroll,WriteString,WriteLn,WriteCard,Write,
                       WriteReal,DibujaCuadro;
  FROM Strings IMPORT  CompareStr,Delete,Length,Assign,Insert,Concat;
  FROM Geometry IMPORT Rectangle;
  FROM utilerias IMPORT ConvStringToHora,ConvStringToFecha;
  FROM LISTAS IMPORT
                       TiempoSeg,TiempoNoPerPtr,TiempoPtr,
                       MensajePtr,PrimerNoPeriodico,PrimerPeriodico,
                       InsertaAntes,BorraLista,BuscaLista1,BuscaLista2,
                       BuscaListaMayor1,BuscaListaMayor2,
                       CreaLista1,CreaLista2,CreaMensaje,InsertaMensaje1,
                       InsertaMensaje2,BorraMensaje1,BorraMensaje2;

  FROM TECLADO IMPORT  StartDevice, StopDevice;    (* tres manejadores  *)
  FROM TIMER IMPORT    ComienzaTimer,ParaTimer,
                       ANIO,MES,DIA,HORA,MIN,SEC; (* de interrupciones *)
  FROM Puerto1 IMPORT  StartReading,StopReading,buffer,BufferSize,out,in,
                       NumDeMen;

  FROM Devices IMPORT  SetDeviceStatus;

  FROM SYSTEM IMPORT CODE,ENABLE;

FROM Procesos IMPORT  CreaProceso, SENAL, ESPERA, ENVIA,Inicializa;

MODULE MENSAJES;
(* exporta lo necesario para administrar los mensajes *)

 IMPORT men1,men,ESPERA,
   mensaje,
   StringToReal, RealToString,
   StringToCard,CardToString,
   GotoXY, Scroll,WriteString,WriteLn,WriteCard,Write,WriteReal,DibujaCuadro
   CompareStr,Delete,Length,Assign,
   Rectangle,
   ConvStringToHora,ConvStringToFecha,
   TiempoSeg,TiempoNoPerPtr,TiempoPtr,
   MensajePtr,PrimerNoPeriodico,PrimerPeriodico,
   InsertaAntes,BorraLista,BuscaLista1,BuscaLista2,
```

```
    BuscaListaMayor1,BuscaListaMayor2,
    CreaLista1,CreaLista2,CreaMensaje,InsertaMensaje1,
    InsertaMensaje2,BorraMensaje1,BorraMensaje2,ANIO,MES,DIA,HORA,MIN,SEC,
    ConvFechaADias,ConvHoraAHoras,SumaFechaHora;

    EXPORT MenuMensajes,Rec;                    -|

    PROCEDURE VerifHora(VAR DIA,MES,ANIO,HORA,MIN,SEC : CARDINAL;
                                                TiempoEspera : CARDINAL);
    (* este procedimiento se usa para sumar el tiempo de espera antes
       de ver si ya recibimos contestacion a un mensaje, verifica que
       se realizen los cambios necesarios en la fecha y hora resultante *)
    BEGIN

        IF (SEC+TiempoEspera) >= 60  THEN
          SEC := (SEC+TiempoEspera) MOD 60;INC(MIN);
          IF MIN = 60  THEN
            MIN:=0;INC(HORA);
            IF HORA = 24 THEN
              HORA:=0;INC(DIA);
              IF (DIA = 29) AND (MES = 3)  THEN
                 DIA:= 1;INC(MES);
                 IF MES = 13 THEN
                   MES := 1; INC(ANIO);
                 END;
              ELSIF
                 (DIA = 31) AND (MES IN {4,6,9,11}) THEN
                   DIA := 1; INC(MES);
                   IF MES = 13 THEN
                     MES := 1; INC(ANIO);
                   END;
              ELSIF
                 (DIA = 32) AND (MES IN {1,3,5,7,8,10,11}) THEN
                   DIA := 1; INC(MES);
                   IF MES = 13 THEN
                     MES := 1; INC(ANIO);
                   END;
              END;  (* if dia *)
            END; (* if hora *)
          END; (* if min *)
        ELSE SEC := SEC + TiempoEspera;
        END; (* if sec *)
    END VerifHora;

    PROCEDURE RecogeBasura() ;
    VAR
      Aux,Aux1      : TiempoNoPerPtr;
      Cabeza        : CHAR;
      horas,mins,secs:ARRAY [0..1] OF CHAR;
      Horas,Sumahoras:REAL;
      sumadias,Sumadias1:CARDINAL;
      Sumadiashoras,Sumadiashoras1:REAL;
      HoraString,MinString,SecString:ARRAY[0..1] OF CHAR;
BEGIN
```

```
    Aux := PrimerNoPeriodico^.Next;
    IF Aux <> PrimerNoPeriodico THEN
         (* convertimos la hora en string para despues
       convertirla a real. no he visto como convertir de
       cardinal a real directamente *)
       CardToString(HORA,horas,2);
       CardToString(MIN,mins,2);
       CardToString(SEC,secs,2);

       ConvHoraAHoras(horas,mins,secs,Horas);
       ConvFechaADias(DIA,MES,ANIO,sumadias);
       SumaFechaHora(sumadias,Horas,Sumadiashoras);

       CardToString(Aux^.Hora,HoraString,2);
       CardToString(Aux^.Min,MinString,2);
       CardToString(Aux^.Sec,SecString,2);

       ConvHoraAHoras(HoraString,MinString,SecString,Sumahoras);
       ConvFechaADias(Aux^.Dia,Aux^.Mes,Aux^.Anio,Sumadias1);
       SumaFechaHora(Sumadias1,Sumahoras,Sumadiashoras1);
   END;
    WHILE (Aux <> PrimerNoPeriodico) AND  (Sumadiashoras1 < Sumadiashoras)
       Aux1 := Aux^.Next;
       IF BorraLista(Aux,1,Cabeza) THEN END;
       Aux := Aux1;
       IF Aux <> PrimerNoPeriodico THEN
          CardToString(Aux^.Hora,HoraString,2);
          CardToString(Aux^.Min,MinString,2);
          CardToString(Aux^.Sec,SecString,2);

          ConvHoraAHoras(HoraString,MinString,SecString,Sumahoras);
          ConvFechaADias(Aux^.Dia,Aux^.Mes,Aux^.Anio,Sumadias1);
          SumaFechaHora(Sumadias1,Sumahoras,Sumadiashoras1);
       END; (* if *)
    END (* while *);

END RecogeBasura;

PROCEDURE FormaMensaje(VAR xmen:mensaje);
  VAR
      Canal:CARDINAL;
BEGIN
    Delete(xmen,0,Length(xmen));
    xmen[0]:='1';  (* el primer caracter del mensaje indica el destino *)
    Scroll(Rec,0);
    GotoXY(3,5);
    WriteString("Mensajes Para Procesador de Comunicaciones");
    GotoXY(2,7);
    WriteString("1.- Valor de todas las variables de UTR ");
    GotoXY(2,8);
    WriteString("2.- Alta a un PC ");
    GotoXY(2,9);
    WriteString("3.- Baja a un PC ");
    GotoXY(2,10);
```

```
      WriteString("4.- Modifica un PC ");
      GotoXY(2,11);
      Correcto := FALSE;
      WHILE NOT(Correcto) DO
        GotoXY(2,18);
        WriteString("Seleccione Opcion --|---> ");
        ESPERA(men1);StringToCard(men,Opcion,Busy);
        IF (Opcion > 0) AND (Opcion <= 1) THEN
          Correcto := TRUE;
        END; (* if *)
      END; (* While *)
      CASE Opcion OF
        1:
           menaver[1]:= CHR(21);
           xmen[1]:= CHR(22);      (* syn *)
           xmen[2]:= CHR(22);      (* syn *)
           xmen[3]:= CHR(21);      (* codigo de operacion *)
           Correcto := FALSE:
           WHILE NOT(Correcto) DO
            GotoXY(2,18);
            WriteString("# de Canal <1,2,3,4>?  ------> ");
            ESPERA(men1);StringToCard(men,Canal,Busy);
            IF (Canal > 0) AND (Canal <= 1) THEN
              Correcto := TRUE;
            END; (* if *)
           END; (* While *)
           xmen[4]:= CHR(Canal);
           menaver[2] := CHR(Canal);
           Correcto:= FALSE;
           WHILE NOT(Correcto) DO
            GotoXY(2,18);
            WriteString("# de UTR <1,2>?    ----------> ");
            ESPERA(men1);StringToCard(men,Canal,Busy);
            IF (Canal > 0) AND (Canal <= 2) THEN
              Correcto := TRUE;
            END; (* if *)
           END; (* While *)
           xmen[5]:= CHR(Canal);
           menaver[3] := CHR(Canal);
           menaver[4] := CHR(0);      (* delimita el mensaje  *)
           Canal := 5;
       |2: (* No Instrumentada *)

       |3: (* No Instrumentada *)

       |4: (* No Instrumentada *)

      END (* case *);
      Delete(xmen,Canal+1,(Length(xmen)-Canal));
END FormaMensaje;

PROCEDURE  InsertaMensajePeriodico1();
BEGIN
  Scroll(Rec,0);
  GotoXY(6,5);
```

```
    WriteString(" Insertar Mensaje Periodico Personal ");
    GotoXY(2,8);
    WriteString(" Mensaje--> ");
    ESPERA(men1); Assign(men,Men);
    Correcto := FALSE;
    WHILE NOT(Correcto) DO
      GotoXY(2,11);
      WriteString("Tiempo en Segundos--> ");
      ESPERA(men1); Assign(men,tiempo);
      StringToReal(tiempo,TiempoReal,Done);
      IF (ABS(TiempoReal) > 0.0) AND (ABS(TiempoReal) <= 86400.0)
       THEN
        Correcto := TRUE;
      END (* if *);
    END; (* while *)
    InsertaMensaje2(Men,TiempoReal);
END InsertaMensajePeriodico1;
PROCEDURE InsertaMensajePeriodico2() ;
  VAR
       xmen:mensaje;
BEGIN
  FormaMensaje(xmen);
  Scroll(Rec,0);
  Correcto := FALSE;
  WHILE NOT(Correcto) DO
    GotoXY(2,11);
    WriteString("Tiempo en Segundos--> ");
    ESPERA(men1);Assign(men,tiempo);
    StringToReal(tiempo,TiempoReal,Done);
    IF (ABS(TiempoReal) > 0.0) AND (ABS(TiempoReal) <= 86400.0)
     THEN
      Correcto := TRUE;
    END (* if *);
  END; (* while *)
  InsertaMensaje2(xmen,TiempoReal);
   CASE ORD(menaver[1]) OF
     21 : InsertaMensaje2(menaver,TiempoReal + 5.0);
   |  4  :
     END (* Case *);
END InsertaMensajePeriodico2;

PROCEDURE InsertaMensajePeriodico() ;
BEGIN
  Opcion:= 0;
  WHILE Opcion <> 3 DO
    Scroll(Rec,0);
    GotoXY(6,5);
    WriteString("Insercion De Mensajes Periodicos");
    GotoXY(2,7);
    WriteString("1.- Mensaje Personal ");
    GotoXY(2,8);
    WriteString("2.- Mensaje a Procesador de Comunicaciones ");
    GotoXY(2,9);
    WriteString("3.- Regresar a Menu Mensajes");
    Correcto := FALSE;
```

```
    WHILE NOT(Correcto) DO
      GotoXY(2,11);
      WriteString("Seleccione Opcion ------> ");
      ESPERA(men1);StringToCard(men,Opcion,Busy);
      IF (Opcion > 0) AND (Opcion <= 3) THEN
        Correcto := TRUE;
      END; (* if *)
    END; (* While *)
    CASE Opcion OF
      1:
        InsertaMensajePeriodico1();
     |2:
        InsertaMensajePeriodico2();
     |3:


        (* implementar salida a menu principal *)

    END (* case *);

  END; (* While *)
END InsertaMensajePeriodico;

PROCEDURE BorraMensajePeriodico1();
BEGIN
  Scroll(Rec,0);
  GotoXY(10,5);
  WriteString(" Borrar Mensaje Periodico ");
  GotoXY(2,8);
  WriteString("Mensaje a Borrar ---> ");
  ESPERA(men1);Assign(men,Men);
  Correcto := FALSE;
  WHILE NOT(Correcto) DO
    GotoXY(2,11);
    WriteString("Tiempo en Segundos ---->");
    ESPERA(men1); Assign(men,tiempo);
    StringToReal(tiempo,TiempoReal,Done);
    IF (ABS(TiempoReal) > 0.0) AND (ABS(TiempoReal) <= 86400.0)
     THEN
      Correcto := TRUE;
    END (* if *);
  END (* while *);
  IF BorraMensaje2(Men,TiempoReal) THEN
  END;
END BorraMensajePeriodico1;
PROCEDURE BorraMensajePeriodico2() ;
  VAR
    xmen:mensaje;
BEGIN
  FormaMensaje(xmen);
  Scroll(Rec,0);
  Correcto := FALSE;
  WHILE NOT(Correcto) DO
    GotoXY(2,11);
    WriteString("Tiempo en Segundos--> ");
```

```
      ESPERA(men1);Assign(men,tiempo);
      StringToReal(tiempo,TiempoReal,Done);
      IF (ABS(TiempoReal) > 0.0) AND (ABS(TiempoReal) <= 86400.0)
       THEN
        Correcto := TRUE;
      END (* if *);
    END; (* while *)
    IF BorraMensaje2(xmen,TiempoReal) THEN
    END;
    CASE ORD(menaver[1]) OF
       21 : IF BorraMensaje2(menaver,TiempoReal + 5.0) THEN END;
    |  4  :
       END (* Case *);
END BorraMensajePeriodico2;

  PROCEDURE BorraMensajePeriodico() ;
  BEGIN
      Opcion:= 0;
      WHILE Opcion <> 3 DO
        Scroll(Rec,0);
        GotoXY(5,5);
        WriteString(" Menu De Borrado De Mensajes Periodicos");
        GotoXY(2,7);
        WriteString("1.- Mensaje Personal ");
        GotoXY(2,8);
        WriteString("2.- Mensaje a Procesador de Comunicaciones ");
        GotoXY(2,9);
        WriteString("3.- Regresar a Menu Mensajes");
        Correcto := FALSE;
        WHILE NOT(Correcto) DO
          GotoXY(2,11);
          WriteString("Seleccione Opcion ------> ");
          ESPERA(men1);StringToCard(men,Opcion,Busy);
          IF (Opcion > 0) AND (Opcion <= 3) THEN
            Correcto := TRUE;
          END; (* if *)
        END; (* While *)
        CASE Opcion OF
          1:
            BorraMensajePeriodico1();
         |2:
            BorraMensajePeriodico2();
         |3:


          (* implementar salida a menu principal *)

        END (* case *);

      END; (* While *)
  END BorraMensajePeriodico;


  PROCEDURE InsertaMensajeNoPeriodico1() ;
   VAR
```

```
    hh,mm,ss,dd,mes,aa:CARDINAL;
    Fecha,hora:ARRAY[0..7] OF CHAR;
BEGIN
  Scroll(Rec,0);
  GotoXY(10,5);
  WriteString(" Insertar Mensaje No Periodico ");
  GotoXY(2,8);
  WriteString("Mensaje a Insertar---> ");
  ESPERA(men1);Assign(men,Men);
  Correcto := FALSE;
    GotoXY(2,11);
    WriteString("Dame Fecha  <dd/mm/aa>    --> ");
    REPEAT
      WriteString("           ");
      GotoXY(28,11);
      ESPERA(men1);Assign(men,Fecha);
      IF ConvStringToFecha(Fecha,dd,mes,aa) THEN
        Correcto := TRUE
      END; (* if str to fecha *)
    UNTIL Correcto = TRUE ;

    Correcto := FALSE;
    GotoXY(2,14);
    WriteString(" Dame Hora  <HH:MM:SS>    --> ");
    REPEAT
      WriteString("           ");
      GotoXY(28,14);
      ESPERA(men1);Assign(men,hora);
      IF ConvStringToHora(hora,hh,mm,ss) THEN
        Correcto:=TRUE;
      END;(*if str to hora*)
    UNTIL Correcto= TRUE;
    InsertaMensaje1(Men,dd,mes,aa,hh,mm,ss);
END InsertaMensajeNoPeriodico1;

PROCEDURE InsertaMensajeNoPeriodico2() ;
  VAR
      xmen:mensaje;
   hh,mm,ss,dd,mes,aa:CARDINAL;
   Fecha,hora:ARRAY[0..7] OF CHAR;
BEGIN
    FormaMensaje(xmen);
    Correcto := FALSE;
    GotoXY(2,11);
    WriteString("Dame Fecha  <dd/mm/aa>    --> ");
    REPEAT
      WriteString("           ");
      GotoXY(28,11);
      ESPERA(men1);Assign(men,Fecha);
      IF ConvStringToFecha(Fecha,dd,mes,aa) THEN
        Correcto := TRUE
      END; (* if str to fecha *)
    UNTIL Correcto = TRUE ;

    Correcto := FALSE;
```

```
      GotoXY(2,14);
      WriteString(" Dame Hora  <HH:MM:SS>    --> ");
      REPEAT
        WriteString("           ");
        GotoXY(28,14);
        ESPERA(men1);Assign(men,hora);  -|
        IF ConvStringToHora(hora,hh,mm,ss) THEN
          Correcto:=TRUE;
        END;(*if str to hora*)
      UNTIL Correcto= TRUE;
      InsertaMensaje1(xmen,dd,mes,aa,hh,mm,ss);
      CASE ORD(menaver[1]) OF
       21 :VerifHora(dd,mes,aa,hh,mm,ss,5);
           InsertaMensaje1(menaver,dd,mes,aa,hh,mm,ss);
    |  4  :
       END (* Case *);
END InsertaMensajeNoPeriodico2;

PROCEDURE InsertaMensajeNoPeriodico() ;
BEGIN
  Opcion := 0;
  WHILE Opcion <> 3 DO
    Scroll(Rec,0);
    GotoXY(5,5);
    WriteString("Insercion De Mensajes No Periodicos");
    GotoXY(2,7);
    WriteString("1.- Mensaje Personal ");
    GotoXY(2,8);
    WriteString("2.- Mensaje a Procesador de Comunicaciones ");
    GotoXY(2,9);
    WriteString("3.- Regresar a Menu Mensajes");
    Correcto := FALSE;
    WHILE NOT(Correcto) DO
      GotoXY(2,11);
      WriteString("Seleccione Opcion -----> ");
      ESPERA(men1);StringToCard(men,Opcion,Busy);
      IF (Opcion > 0) AND (Opcion <= 3) THEN
        Correcto := TRUE;
      END; (* if *)
    END; (* While *)
    CASE Opcion OF
      1:
        InsertaMensajeNoPeriodico1();
     |2:
        InsertaMensajeNoPeriodico2();
     |3:

        (* implementar salida a menu principal *)

    END (* case *);

  END; (* While *)
END InsertaMensajeNoPeriodico;
```

```
PROCEDURE BorraMensajeNoPeriodico1() ;
  VAR
     hh,mm,ss,dd,mes,aa:CARDINAL;
     Fecha,hora:ARRAY[0..7] OF CHAR;
BEGIN
  Scroll(Rec,0);
  GotoXY(10,5);
  WriteString(" Borrar Mensaje No Periodico ");
  GotoXY(2,8);
  WriteString(" Dame Mensaje a Borrar ----> ");
  ESPERA(men1); Assign(men,Men);
  Correcto := FALSE;
    GotoXY(2,11);
    WriteString("Dame Fecha  <dd/mm/aa>    --> ");
    REPEAT
      WriteString("           ");
      GotoXY(28,11);
      ESPERA(men1);Assign(men,Fecha);
      IF ConvStringToFecha(Fecha,dd,mes,aa) THEN
        Correcto := TRUE
      END; (* if str to fecha *)
    UNTIL Correcto = TRUE ;

    Correcto := FALSE;
    GotoXY(2,14);
    WriteString(" Dame Hora  <HH:MM:SS>    --> ");
    REPEAT
      WriteString("           ");
      GotoXY(28,14);
      ESPERA(men1);Assign(men,hora);
      IF ConvStringToHora(hora,hh,mm,ss) THEN
        Correcto:=TRUE;
      END;(*if str to hora*)
    UNTIL Correcto= TRUE;
    IF BorraMensaje1(Men,dd,mes,aa,hh,mm,ss) THEN
      GotoXY(2,20);
      WriteString(" Mensaje Borrado ");
    ELSE
      GotoXY(2,20);
      WriteString(" No existe mensaje ");
    END;

END BorraMensajeNoPeriodico1;

PROCEDURE BorraMensajeNoPeriodico2() ;
  VAR
     xmen:mensaje;
     hh,mm,ss,dd,mes,aa:CARDINAL;
     Fecha,hora:ARRAY[0..7] OF CHAR;
BEGIN
  FormaMensaje(xmen);
    Correcto := FALSE;
    GotoXY(2,11);
    WriteString("Dame Fecha  <dd/mm/aa>    --> ");
    REPEAT
```

```
        WriteString("           ");
        GotoXY(28,11);
        ESPERA(men1); Assign(men,Fecha);
        IF ConvStringToFecha(Fecha,dd,mes,aa) THEN
          Correcto := TRUE
        END; (* if str to fecha *)
      UNTIL Correcto = TRUE ;

      Correcto := FALSE;
      GotoXY(2,14);
      WriteString(" Dame Hora  <HH:MM:SS>   --> ");
      REPEAT
        WriteString("           ");
        GotoXY(28,14);
        ESPERA(men1);Assign(men,hora);
        IF ConvStringToHora(hora,hh,mm,ss) THEN
          Correcto:=TRUE;
        END;(*if str to hora*)
      UNTIL Correcto= TRUE;
      IF BorraMensaje1(xmen,dd,mes,aa,hh,mm,ss) THEN
        GotoXY(2,20);
        WriteString(" Mensaje Borrado ");
      ELSE
        GotoXY(2,20);
        WriteString(" No existe mensaje ");
      END;
      CASE ORD(menaver[1]) OF
       21 :VerifHora(dd,mes,aa,hh,mm,ss,5);
           IF BorraMensaje1(menaver,dd,mes,aa,hh,mm,ss) THEN
            END;
    |  4  :
       END (* Case *);

END BorraMensajeNoPeriodico2;

PROCEDURE BorraMensajeNoPeriodico() ;
BEGIN
  Opcion:= 0;
  WHILE Opcion <> 3 DO
    Scroll(Rec,0);
    GotoXY(4,5);
    WriteString(" Menu De Borrado De Mensajes No Periodicos");
    GotoXY(2,7);
    WriteString("1.- Mensaje Personal ");
    GotoXY(2,8);
    WriteString("2.- Mensaje a Procesador de Comunicaciones ");
    GotoXY(2,9);
    WriteString("3.- Regresar a Menu Mensajes");
    Correcto := FALSE;
    WHILE NOT(Correcto) DO
      GotoXY(2,11);
      WriteString("Seleccione Opcion ------> ");
      ESPERA(men1);StringToCard(men,Opcion,Busy);
      IF (Opcion > 0) AND (Opcion <= 3) THEN
        Correcto := TRUE;
```

```
        END; (* if *)
      END; (* While *)
      CASE Opcion OF
        1:
          BorraMensajeNoPeriodico1();
       |2:                                    -|
          BorraMensajeNoPeriodico2();
       |3:

          (* implementar salida a menu principal *)

      END (* case *);

    END; (* While *)

  END BorraMensajeNoPeriodico;


  PROCEDURE ListaTiemposPeriodicos() ;
     VAR
         Aux     : TiempoPtr;
         x,y : CARDINAL;
  BEGIN
    Scroll(Rec,0);
    x := 5;  y := 4;
    Aux := PrimerPeriodico^.Next;
    IF Aux = PrimerPeriodico THEN
      GotoXY(6,19);
      WriteString("No Hay Tiempos Periodicos Programados");
    ELSE
      GotoXY(4,3);
      WriteString("REPORTE DE TIEMPOS PERIODICOS ");
      WHILE Aux <> PrimerPeriodico DO
        GotoXY(y,x);
        WriteReal(Aux^.TiempoPer,2,10);
        x := x + 1;
        IF x = 20 THEN  y := 25; x := 5; END;
        Aux := Aux^.Next;
      END (* while *);
    END (* if *);
    GotoXY(2,20);
    WriteString("Para Volver al Menu Principal Presione <ret>");
    ESPERA(men1);
  END ListaTiemposPeriodicos;

  PROCEDURE ListaTiemposNoPeriodicos() ;
    VAR
        Aux     : TiempoNoPerPtr;
        x,y,y1 : CARDINAL;
  BEGIN
    RecogeBasura;
    Scroll(Rec,0);
    x := 5; y:= 10; y1:=25;
    Aux := PrimerNoPeriodico^.Next;
    IF Aux = PrimerNoPeriodico THEN
```

```
      GotoXY(5,19);
      WriteString(" NO hay Tiempos No Periodicos programados ");
    ELSE
      GotoXY(5,3);
      WriteString("REPORTE DE TIEMPOS NO PERIODICOS ");
      WHILE Aux <> PrimerNoPeriodico DO
        GotoXY(y,x);
        WriteCard(Aux^.Dia,2);
        Write('/');
        WriteCard(Aux^.Mes,2);
        Write('/');
        WriteCard(Aux^.Anio,2);
        GotoXY(y1,x);
        WriteCard(Aux^.Hora,2);
        Write(':');
        WriteCard(Aux^.Min,2);
        Write(':');
        WriteCard(Aux^.Sec,2);
        x:=x+1;
        IF x = 20 THEN x:=5; y:= 30; y1:=40; END;
        Aux := Aux^.Next;
      END (* while *);
    END (* if *);
    GotoXY(2,20);
    WriteString("Para Volver a Menu Presione <ret>");
    ESPERA(men1);
END ListaTiemposNoPeriodicos;

PROCEDURE ListaMensajesPeriodicos ;
  VAR
       Aux    : TiempoPtr;
       AuxMen : MensajePtr;
       x,y,y1 : CARDINAL;
BEGIN
    Scroll(Rec,0);
    x:=5;y:=2;y1:=17;
    Aux := PrimerPeriodico^.Next;
    IF Aux = PrimerPeriodico THEN
      GotoXY(2,19);
      WriteString("No hay tiempos periodicos programados ");
    ELSE
      GotoXY(2,3);
      WriteString("REPORTE DE LOS MENSAJES PERIODICOS");
      GotoXY(2,4);
      WriteString("TIEMPO");
      GotoXY(17,4); WriteString("MENSAJE");
      WHILE Aux <> PrimerPeriodico DO
        AuxMen := Aux^.MenPtr^.Next;
        WHILE AuxMen <> Aux^.MenPtr DO
          GotoXY(y,x);
          WriteReal(Aux^.TiempoPer,2,10);
          GotoXY(y1,x);
          WriteString(AuxMen^.Men);
          x := x+1;
          IF x = 20 THEN y:= 35;y1:=40;x:=5; END;
```

```
        AuxMen := AuxMen^.Next;
      END (* while *);
      Aux := Aux^.Next;
    END (* while *);
  END (* if *);
  GotoXY(2,20);
  WriteString("Para Volver a Menu Presione <ret>");
  ESPERA(men1);
END ListaMensajesPeriodicos;

PROCEDURE ListaMensajesNoPeriodicos();
  VAR
      Aux        : TiempoNoPerPtr;
      AuxMen     : MensajePtr;
      x,y,y1,y2 : CARDINAL;
BEGIN
  RecogeBasura;
  Scroll(Rec,0);
  x := 5;  y := 3;  y1 := 17;  y2:= 35;
  Aux := PrimerNoPeriodico^.Next;
  IF Aux = PrimerNoPeriodico THEN
    GotoXY(2,19);
    WriteString("No Hay tiempos no periodicos programados ");
  ELSE
    GotoXY(2,3);
    WriteString("REPORTE DE MENSAJES NO PERIODICOS");
    GotoXY(7,4); WriteString("FECHA");
    GotoXY(17,4); WriteString("HORA");
    GotoXY(30,4); WriteString("MENSAJE");
    WHILE Aux <> PrimerNoPeriodico DO
      AuxMen := Aux^.MenPtr^.Next;
      WHILE AuxMen <> Aux^.MenPtr DO
        GotoXY(y,x);
        WriteCard(Aux^.Dia,2);
        Write('/');
        WriteCard(Aux^.Mes,2);
        Write('/');
        WriteCard(Aux^.Anio,2);
        GotoXY(y1,x);
        WriteCard(Aux^.Hora,2);
        Write(':');
        WriteCard(Aux^.Min,2);
        Write(':');
        WriteCard(Aux^.Sec,2);
        GotoXY(y2,x);
        WriteString(AuxMen^.Men);
        x:=x+1;
        IF x = 20 THEN x:=5; y:= 3; y1:=17; y2:= 35;
        GotoXY(2,19);WriteString("Mas Presione <a><ret>");ESPERA(men1);END
        AuxMen := AuxMen^.Next;
      END (* while *);
      Aux := Aux^.Next;
    END (* while *);
  END (* if *);
  GotoXY(2,20);
```

```
   WriteString("Para Volver a Menu  Presione <a> <ret>");
   ESPERA(men1);
END ListaMensajesNoPeriodicos;

PROCEDURE ListaUnTiempoPeriodico() ;
    VAR                                    -|
        ListaiPtr : TiempoPtr;
        MensPoint : MensajePtr;
        x,y       :CARDINAL;
BEGIN
 Scroll(Rec,0);
 x:=5; y:=15;
 Correcto := FALSE;
  WHILE NOT(Correcto) DO
    GotoXY(2,11);
    WriteString(" Dame Tiempo en Segundos ---->");
    ESPERA(men1);
    Assign(men,tiempo);
    StringToReal(tiempo,TiempoReal,Done);
    IF (ABS(TiempoReal) > 0.0) AND (ABS(TiempoReal) <= 86400.0)
     THEN
      Correcto := TRUE;
    END (* if *);
  END (* while *);
   IF BuscaLista2(TiempoReal,ListaiPtr) THEN
     Scroll(Rec,0);
     GotoXY(2,2);
     WriteString(" REPORTE DE MENSAJES EXISTENTES ");
     GotoXY(2,4);
     WriteString("mensaje en lista  ");
     MensPoint := ListaiPtr^.MenPtr^.Next;
     WHILE  (MensPoint <> ListaiPtr^.MenPtr) DO
       GotoXY(y,x);
       WriteString(MensPoint^.Men);
       x := x+1;
       IF x = 20 THEN x:=5;y:=30; END;
       MensPoint := MensPoint^.Next;
     END; (* while *)
    ELSE
      GotoXY(2,19);
      WriteString("No Existen mensajes en ese tiempo ");
    END;
    GotoXY(2,20);
    WriteString(" Presione <a> <ret> ");
    ESPERA(men1);
END ListaUnTiempoPeriodico;

PROCEDURE ListaUnTiempoNoPeriodico() ;
   VAR
       Aux    : TiempoNoPerPtr;
       AuxMen : MensajePtr;
       Fecha  : ARRAY[0..7] OF CHAR;
       hora   : ARRAY[0..7] OF CHAR;
     x,y,dd,mes,aa,hh,mm,ss:CARDINAL;
BEGIN
```

```
  Scroll(Rec,0);
  x:=5;y:=15;
  Correcto := FALSE;
    GotoXY(2,11);
    WriteString("Dame Fecha  <dd/mm/aa>    --> ");
    REPEAT
      WriteString("           ");
      GotoXY(28,11);
      ESPERA(men1);
      Assign(men,Fecha);
      IF ConvStringToFecha(Fecha,dd,mes,aa) THEN
        Correcto := TRUE
      END; (* if str to fecha *)
    UNTIL Correcto = TRUE ;

    Correcto := FALSE;
    GotoXY(2,14);
    WriteString(" Dame Hora  <HH:MM:SS>    --> ");
    REPEAT
      WriteString("           ");
      GotoXY(28,14);
      ESPERA(men1);
      Assign(men,hora);
      IF ConvStringToHora(hora,hh,mm,ss) THEN
        Correcto:=TRUE;
      END;(*if str to hora*)
    UNTIL Correcto= TRUE;
    Scroll(Rec,0);
    IF BuscaLista1(dd,mes,aa,hh,mm,ss,Aux) THEN
      GotoXY(6,3);
      WriteString("REPORTE DE MENSAJES NO PERIODICOS");
      AuxMen := Aux^.MenPtr^.Next;
      WHILE AuxMen <> Aux^.MenPtr DO
        GotoXY(y,x);
        WriteString(AuxMen^.Men);
        x := x+1;
        IF x = 20 THEN y:= 15; x:= 5; END;
        AuxMen := AuxMen^.Next;
      END (* while *);
    ELSE
      GotoXY(2,19);
      WriteString(" no hay mensajes en esa fecha y hora ");
    END; (* if *)
    GotoXY(2,20);
    WriteString(" Presione cualquier tecla para volver a menu principal ")
    ESPERA(men1);
END ListaUnTiempoNoPeriodico;
```

```
PROCEDURE MenuMensajes();
BEGIN
  Opcion := 0;
  WHILE Opcion <> 11 DO
    Scroll(Rec,0);
    GotoXY(10,5);
    WriteString(" Menu De Mensajes ");
    GotoXY(2,7);
    WriteString("1.- Insertar Mensaje Periodico");
    GotoXY(2,8);
    WriteString("2.- Borrar Mensaje Periodico");
    GotoXY(2,9);
    WriteString("3.- Insertar Mensaje No Periodico");
    GotoXY(2,10);
    WriteString("4.- Borrar Mensaje No  Periodico");
    GotoXY(2,11);
    WriteString("5.- Listar Tiempos Periodicos");
    GotoXY(2,12);
    WriteString("6.- Listar Tiempos No Periodicos");
    GotoXY(2,13);
    WriteString("7.- Listar Mensajes de tiempo periodicos");
    GotoXY(2,14);
    WriteString("8.- Listar Mensajes de tiempos no periodicos");
    GotoXY(2,15);
    WriteString("9.- Listar Mensajes de un tiempo periodico");
    GotoXY(2,16);
    WriteString("10.- Listar Mensajes de un tiempo no periodico");
    GotoXY(2,17);
    WriteString("11 .- Menu Principal ");
    Correcto := FALSE;
    WHILE NOT(Correcto) DO
      GotoXY(2,19);
      WriteString("Seleccione Opcion ------> ");
      ESPERA(men1); StringToCard(men,Opcion,Busy);
      IF (Opcion > 0) AND (Opcion <= 11) THEN
        Correcto := TRUE;
      END; (* if *)
    END; (* While *)
    CASE Opcion OF
      1:
        InsertaMensajePeriodico();
     |2:
        BorraMensajePeriodico();
     |3:
        InsertaMensajeNoPeriodico();
     |4:
        BorraMensajeNoPeriodico();
     |5:
        ListaTiemposPeriodicos();
     |6:
        ListaTiemposNoPeriodicos();
     |7:
        ListaMensajesPeriodicos();
     |8:
        ListaMensajesNoPeriodicos();
```

```
        |9:
           ListaUnTiempoPeriodico();
        |10:
           ListaUnTiempoNoPeriodico();
        |11:
           (* implementar salida a menu principal *)

      END (* case *);

    END; (* While *)

  END MenuMensajes;
  VAR
       tiempo :ARRAY[0..15] OF CHAR;
       Opcion,Hora,min,seg:CARDINAL;
       Correcto,Done :BOOLEAN;
       Rec:Rectangle;
       menaver,Men:mensaje;
       TiempoReal:REAL;
       Busy:BOOLEAN;

BEGIN
  Opcion:=0;Delete(menaver,0,Length(menaver));menaver[0] := '5';
END MENSAJES;

PROCEDURE InterpretaMensajePC();
(*************************************************************)
(* proceso que interpreta los mensajes enviados por el
procesador de comunicaciones *)
(*************************************************************)


PROCEDURE POSAUX();
(* posiciona el elevador *)

BEGIN
    IF Estado = Parado THEN
        CASE Piso OF
          1:Screen[19,Renx].char := car;
            Screen[19,Renx + 1].char := car;
            IF Elev = 11 THEN ulposx1 := 19 ELSE ulposx2 := 19 END;
       |    2:
            Screen[16,Renx].char := car;
            Screen[16,Renx + 1].char := car;
            IF Elev = 11 THEN ulposx1 := 16 ELSE ulposx2 := 16 END;
       |    3:
            Screen[13,Renx].char := car;
            Screen[13,Renx + 1].char := car;
            IF Elev = 11 THEN ulposx1 := 13 ELSE ulposx2 := 13 END;
       |    4:
            Screen[10,Renx].char := car;
            Screen[10,Renx + 1].char := car;
            IF Elev = 11 THEN ulposx1 := 10 ELSE ulposx2 := 10 END;
       |    5:
            Screen[7,Renx].char := car;
```

```
          Screen[7,Renx + 1].char := car;
          IF Elev = 11 THEN ulposx1 := 7 ELSE ulposx2 := 7 END;
    |     6 :
            Screen[17,Renx].char := car;
            Screen[17,Renx + 1].char := car;
            IF Elev = 11 THEN ulposx1 := 17 ELSE ulposx2 := 17 END;
    |     7:
            Screen[14,Renx].char := car;
            Screen[14,Renx + 1].char := car;
            IF Elev = 11 THEN ulposx1 := 14 ELSE ulposx2 := 14 END;
    |     8:
          Screen[11,Renx].char := car;
          Screen[11,Renx + 1].char := car;
          IF Elev = 11 THEN ulposx1 := 11 ELSE ulposx2 := 11 END;
    |     9:
            Screen[8,Renx].char := car;
            Screen[8,Renx + 1].char := car;
            IF Elev = 11 THEN ulposx1 := 8 ELSE ulposx2 := 8 END;
      ELSE
      END (* case Piso *);
ELSE (* estado <> parado *)
      CASE Piso OF
         1:Screen[19,Renx].char := car;
            Screen[19,Renx + 1].char := ' ';
            IF Elev = 11 THEN ulposx1 := 19 ELSE ulposx2 := 19 END;
      |   2:
            Screen[16,Renx].char := car;
            Screen[16,Renx + 1].char := ' ';
            IF Elev = 11 THEN ulposx1 := 16 ELSE ulposx2 := 16 END;
      |   3:
            Screen[13,Renx].char := car;
            Screen[13,Renx + 1].char := ' ';
            IF Elev = 11 THEN ulposx1 := 13 ELSE ulposx2 := 13 END;
      |   4:
            Screen[10,Renx].char := car;
            Screen[10,Renx + 1].char := ' ';
            IF Elev = 11 THEN ulposx1 := 10 ELSE ulposx2 := 10 END;
      |   5:
            Screen[7,Renx].char := car;
            Screen[7,Renx + 1].char := ' ';
            IF Elev = 11 THEN ulposx1 := 7 ELSE ulposx2 := 7 END;
      |   6 :
            Screen[17,Renx].char := car;
            Screen[17,Renx + 1].char := ' ';
            IF Elev = 11 THEN ulposx1 := 17 ELSE ulposx2 := 17 END;
      |   7:
            Screen[14,Renx].char := car;
            Screen[14,Renx + 1].char := ' ';
            IF Elev = 11 THEN ulposx1 := 14 ELSE ulposx2 := 14 END;
      |   8:
          Screen[11,Renx].char := car;
          Screen[11,Renx + 1].char := ' ';
          IF Elev = 11 THEN ulposx1 := 11 ELSE ulposx2 := 11 END;
      |   9:
            Screen[8,Renx].char := car;
```

```
                Screen[8,Renx + 1].char := ' ';
                IF Elev = 11 THEN ulposx1 := 8 ELSE ulposx2 := 8 END;
            ELSE
            END (* case Piso *);
        END (* if Estado <> parado *);

    END POSAUX;


PROCEDURE PosicionaElev();
(* posiciona el elevador *)
 BEGIN
    CASE Elev OF
      11 :
      Screen[ulposx1,58].char := ' ';
      Screen[ulposx1,59].char :=  ' ';
      Renx := 58;
      POSAUX;
    |12:
       Screen[ulposx2,73].char := ' ';
       Screen[ulposx2,74].char := ' ';
       Renx := 73;
       POSAUX;
   ELSE
   END (* case *);
  END PosicionaElev;


  PROCEDURE Alarma(Tipo:CARDINAL);
  (* envia un mensaje de alarma al video *)

   VAR
    MenAlarma : ARRAY[0..60] OF CHAR;
    ElevStr   : ARRAY[0..3] OF CHAR;
    x         : CARDINAL;
  BEGIN
    Delete(MenAlarma,0,Length(MenAlarma));
    CASE Tipo OF
     1 : Concat(MenAlarma,"Elevador   Subiendo y Puerta Abierta",MenAlarma);
   | 2 : Concat(MenAlarma,"Elevador   Subiendo y Puerta Abriendo",MenAlarma);
   | 3 : Concat(MenAlarma,"Elevador   Subiendo y Puerta Cerrando",MenAlarma);
   | 4 : Concat(MenAlarma,"Elevador   Subiendo y estaba Bajando",MenAlarma);
   | 5 : Concat(MenAlarma,"Elevador   Bajando y Puerta Abierta",MenAlarma);
   | 6 : Concat(MenAlarma,"Elevador   Bajando y Puerta Abriendo",MenAlarma);
   | 7 : Concat(MenAlarma,"Elevador   Bajando y Puerta Cerrando",MenAlarma);
   | 8 : Concat(MenAlarma,"Elevador   Bajando y estaba Subiendo",MenAlarma);
   | 9 : Concat(MenAlarma,"Elevador   Parado entre 2 pisos",MenAlarma);
   |10 : Concat(MenAlarma,"Elevador   con Puerta Abierta entre pisos",MenAlar
ma);
   |11 : Concat(MenAlarma,"Elevador   con Puerta Abriendo entre 2 pisos",MenA
ma);
   |12 : Concat(MenAlarma,"Elevador   'Pasajero con Problemas*'",MenAlarma);
   ELSE
    END (* case *);
   CardToString(Elev,ElevStr,2);
   Insert(ElevStr,MenAlarma,9);
```

```
  FOR x:=4 TO 79 DO
    Screen[23,x].char := ' ';
  END;
  FOR x:=0 TO Length(MenAlarma) DO          (* escribe el mensaje *)
    Screen[23,x+4].char := MenAlarma[x];    (* de Alarma en pantalla *)
  END; (* for *)

END Alarma;


VAR
    CODIGO         : BITSET;
    Estado         : (Subiendo,Bajando,Parado);
    Puerta         : (Abriendo,Cerrando,Abierta,Cerrada);
    Piso           : [1..9];
    Error          : CHAR;                  (* indica si hubo o no error *)
    Cerror         : CHAR;                  (* codigo de error si hubo   *)
    CodigoOP       : CHAR;                  (* codigo de operacion       *)
    VarDigitales   : CHAR;                  (* num de variables digitales*)
    VarAnalogicas: CHAR;                    (* num. de var. analogicas   *)
    Canal          : CHAR;                  (* num. de canal             *)
    UTR            : CHAR;                  (* num de utr                *)
    Elev           : CARDINAL;              (* numero de elevador        *)
    x,y            : CARDINAL;              (* auxiliares                *)
    MenDeError     : ARRAY[0..70]OF CHAR;(* en caso de error en mens. *)
    Cod            : ARRAY[0..2] OF CHAR;(* auxiliar                  *)
    car            : CHAR;                  (* representacion del elev.  *)
    Point          : CARDINAL;
    ConDeMen       : CARDINAL;              (* contador de mensajes Puer.*)
BEGIN
  Concat(MenDeError,"Mensaje De    Codigo de OP    con Error tipo   ",MenD
);
  Estado:=Parado; Piso:=2; Puerta:=Cerrada;
  CodigosRecibidos[21,0]:= 0;
  CodigosRecibidos[21,1]:= 0;
  CodigosRecibidos[21,2]:= 0;
  SetDeviceStatus(7,FALSE);out:=0;SetDeviceStatus(7,TRUE);Point := 0;
  LOOP
     ESPERA(men2);
     x:=0;y:=0;
     ConDeMen := NumDeMen;
     CODE(0FAH);NumDeMen := NumDeMen - ConDeMen; CODE(0FBH);
     WHILE ConDeMen <> 0 DO
      REPEAT
       IF Point < 50 THEN  MensajePC[Point] := buffer[out] END;
         IF (MensajePC[Point] = CHR(4)) AND (x = 1) THEN
           y:=5;
         ELSE
           IF MensajePC[Point] = CHR(4)  THEN
             x:=1;
           ELSE
             x:=0;
           END (* if *);
         END (* if *);
       INC(Point);
       SetDeviceStatus(7,FALSE);
```

```
  out := (out + 1) MOD BufferSize;
  SetDeviceStatus(7,TRUE);
UNTIL (y = 5) OR (out = in);
IF y = 5 THEN
Point := 0;
(* decodifica encabezado del mensaje *)
Error    := MensajePC[2];
Cerror   := MensajePC[3];
CodigoOP := MensajePC[4];
IF Error = CHR(6) THEN
  CASE ORD(CodigoOP) OF
    1:
  | 2:
  | 3:
  | 21:
      VarDigitales  := MensajePC[5];
      VarAnalogicas := MensajePC[6];
      Canal         := MensajePC[7];
      UTR           := MensajePC[8];
      x := 0;
      y := 10;

      LOOP                          (* formo el codigo que me indica*)
                                    (* el estado del sistema.       *)
        IF x = 8 THEN               (* si ya son ocho variables     *)
          EXIT;                     (* termina.                     *)
        END;
        IF MensajePC[y] =CHR(1) THEN  (* si la variable es 1 prendo e

         INCL(CODIGO,x);            (* bit correspondiente.          *)
        ELSE
          EXCL(CODIGO,x);
        END;
        y:=y+2;
        INC(x);
      END (* loop *);
      IF (5 IN CODIGO) AND (4 IN CODIGO) THEN  Estado := Bajando
      ELSIF
         (5 IN CODIGO)                       THEN  Estado := Subiendo
      ELSIF
         (4 IN CODIGO)                       THEN  Estado := Parado
      END (* if *);
      IF (3 IN CODIGO) AND (NOT(2 IN CODIGO)) AND
         (NOT(1 IN CODIGO)) AND (NOT(0 IN CODIGO))
                                             THEN  Piso := 5;
      ELSIF
         (NOT(3 IN CODIGO)) AND (2 IN CODIGO) AND
         (1 IN CODIGO) AND (0 IN CODIGO) THEN  Piso := 9;
      ELSIF
         (NOT(3 IN CODIGO)) AND (2 IN CODIGO) AND
         (1 IN CODIGO)                       THEN  Piso := 4;
      ELSIF
         (NOT(3 IN CODIGO)) AND (2 IN CODIGO) AND
         (0 IN CODIGO)                       THEN  Piso := 8;
      ELSIF
         (NOT(3 IN CODIGO)) AND (2 IN CODIGO)
```

```
                                        THEN  Piso := 3;
ELSIF
   (NOT(3 IN CODIGO)) AND (1 IN CODIGO) AND
   (0 IN CODIGO)                        THEN Piso := 7;
ELSIF
   (NOT(3 IN CODIGO)) AND (1 IN CODIGO)
                                        THEN Piso := 2;
ELSIF
   (NOT(3 IN CODIGO)) AND (0 IN CODIGO)
                                        THEN Piso := 6;
ELSIF
   (NOT(3 IN CODIGO))                   THEN Piso := 1;
END; (* if *)

x := ORD(CodigoOP);
CASE ORD(Canal) OF
  1:
  CASE ORD(UTR) OF
     1:
      INC(CodigosRecibidos[21,0]);   (* canal 1*)
      INC(CodigosRecibidos[21,1]);   (* utr   1*)
      Elev := 11;
  | 2:
      INC(CodigosRecibidos[21,0]);   (* canal 1*)
      INC(CodigosRecibidos[21,2]);   (* utr   2*)
     Elev := 12;
  | 3:
  | 5:
    ELSE
  END (* case utr *);

| 2:
| 3:
| 4:
  ELSE
END (* case canal *);
IF  (NOT((Estado = Subiendo) AND (Piso = 5))) AND
    (NOT((Estado = Bajando) AND (Piso = 1 )))  THEN
    CASE Estado OF
        Subiendo :
                   IF NOT(6 IN CODIGO) THEN
                     Alarma(1); car := CHR(178);
                   ELSE
                     car := CHR(24)
                   END;
    |   Bajando  :
                   IF NOT(6 IN CODIGO) THEN
                   Alarma(5); car := CHR(177);
                   ELSE
                     car := CHR(25);
                   END;
    |   Parado   :
                   IF (Piso = 6) OR (Piso = 7) OR
                      (Piso = 8) OR (Piso = 9) THEN
                     Alarma(9);car := CHR(176);
```

```
                                ELSE
                                  car := CHR(219);
                                END;
                        ELSE
                        END (* case estado *);
                        IF (NOT(6 IN CODIGO)) AND ((Piso = 6) OR (Piso = 7) OR
                           (Piso = 8) OR (Piso = 9)) THEN
                           Alarma(10);car := CHR(178)
                        ELSE
                            car := CHR(219)
                        END;
                        IF 7 IN CODIGO THEN Alarma(12) END;
                        PosicionaElev();
                 END (* if *);
                ELSE
             END (* case codigo de operacion *);
          ELSE                                          (********************)
          END (* if *);
          END;(* if y = 5 *)
          DEC(ConDeMen);
       END; (* While condemen <>0 *)
     END (* loop *);
   END InterpretaMensajePC;

   PROCEDURE VerificaRespPC();
   (*****************************************************************)
   (* Proceso que verifica si se recibio respuesta del procesador
de comunicaciones *)
   (*****************************************************************)

    VAR
        tt          : ARRAY[0..30] OF CHAR;
        MenAlarma   : ARRAY[0..70] OF CHAR;
        Busy        : BOOLEAN;
        Cod         : ARRAY[0..2] OF CHAR;
        codigoaux,utraux,x : CARDINAL;
   BEGIN
     Delete(MenAlarma,0,Length(MenAlarma));
     Concat(MenAlarma,"No Recibi Contestacion de    al Codigo de Operacion
enAlarma);
     LOOP
       ESPERA(men3);
       Busy := FALSE;
       codigoaux := ORD(MENSAJEaVER[0]);
       utraux    := ORD(MENSAJEaVER[2]);
       IF (CodigosRecibidos[codigoaux,0] <> 0)       AND
          (CodigosRecibidos[codigoaux,utraux] <> 0) THEN

          DEC(CodigosRecibidos[codigoaux,0]);
          DEC(CodigosRecibidos[codigoaux,utraux]);
          Busy := TRUE;

       END (* if *);

       IF NOT(Busy) THEN
         Delete(MenAlarma,26,2);
```

```
            Delete(MenAlarma,52,2);
            CardToString(ORD(MENSAJEaVER[1]),Cod,1);
            Insert(Cod,MenAlarma,26);
            CardToString(ORD(MENSAJEaVER[2]),Cod,1);
            Insert(Cod,MenAlarma,27);
            CardToString(ORD(MENSAJEaVER[0]),Cod,2);
            Insert(Cod,MenAlarma,52);
            FOR x:=4 TO 79 DO
               Screen[23,x].char := ' ';
            END (* for *);
            FOR x := 0 TO Length(MenAlarma) DO
               Screen[23,x+4].char := MenAlarma[x];
            END (* for *);
          ELSE
            (*tt:="recibicodigo ";
            CardToString(ORD(MENSAJEaVER[1]),Cod,1);
            Concat(tt,Cod,tt);
            CardToString(ORD(MENSAJEaVER[2]),Cod,1);
            Concat(tt,Cod,tt);
            FOR x:=4 TO 79 DO
               Screen[24,x].char := ' ';
            END (* for *);
            FOR x := 0 TO Length(tt) DO
               Screen[24,x+4].char := tt[x];
            END (* for *);  *)
          END; (* if *)
        END (* loop *);
      END VerificaRespPC;


PROCEDURE MenuPri;
(*************************************************************************)
(* Proceso que administra los menus de mensajes *)

  VAR
      Opcion:CARDINAL;

BEGIN
  Rec.left  := 1;
  Rec.top   := 3;
  Rec.right := 50;
  Rec.bottom:= 22;

  LOOP
    Scroll(Rec,0);
    GotoXY(10,0);
    WriteString("MENU PRINCIPAL");
    GotoXY(3,1);
    WriteString("*).- Para ");
    GotoXY(14,1);
    WriteString("3).- Mensajes");
    GotoXY(29,1);
    WriteString("^A BorraA");
    GotoXY(45,1);
    WriteString("^B BorraM");
```

```
    GotoXY(58,1);
    WriteString(" Opcion ---> ");
    ESPERA(men1); StringToCard(men,Opcion,Busy);
    CASE Opcion OF
      1:
    | 2:
    | 3:MenuMensajes;
    | 4:
    | 5:
      ELSE GotoXY(70,1);
    END (*  case *);
  END; (* LOOP *)
END MenuPri;


PROCEDURE Maestro;
(*****************************************************************)
(*  Proceso que saca mensajes de el buffer general,  y activa  al
proceso que le debe dar servicio *)
(*****************************************************************)

  VAR
     ClaveDispositivo:CHAR;
     x,Point:CARDINAL;
BEGIN
  LOOP
    SACA(men);
    ClaveDispositivo:=men[0];
    IF ClaveDispositivo <> '1  THEN
      Delete(men,0,1);
    END;
    IF CompareStr(men,"*") = 0  THEN
      ParaTimer();
      StopReading();
      StopDevice();
      ESPERA(Terminar);
    ELSE
      CASE ClaveDispositivo OF
       '1':
           ENVIA(men2);
     | '2':
     | '3':
     | '4':
           ENVIA(men1);
     | '5':
           MENSAJEaVER := men;
           ENVIA(men3);
     END;
    END (* if *);
  END (* loop *);
END Maestro;
```

```
TYPE
     ScreenType =
            ARRAY[0..24] OF              (* filas *)
            ARRAY[0..79] OF              (* columnas *)
            RECORD char,attr:CHAR END;  (* contenido *)
  VAR
      Screen[0B800H:0H]:ScreenType;

  VAR
     Terminar,men1,men2,men3:SENAL;
     men,MensajePC,MENSAJEaVER : mensaje;
     CodigosRecibidos : ARRAY[0..25] OF
                        ARRAY[0..2] OF CARDINAL;
     Busy             : BOOLEAN;
     ulposx1,ulposx2  : CARDINAL;
     Renx             : [0..78];
BEGIN
  FOR ulposx1 := 0 TO 25 DO                (***************************)
    CodigosRecibidos[ulposx1,0] := 0;      (* Inicialización de      *)
    CodigosRecibidos[ulposx1,1] := 0;      (* la tabla de codigos    *)
    CodigosRecibidos[ulposx1,2] := 0;      (* recibidos              *)
  END (* for *);                          (***************************)

  Rec.left := 53;              (*****************************************)
  Rec.top  := 5;               (*                                       *)
  Rec.right:= 63;              (*                                       *)
  Rec.bottom := 21;            (*                                       *)
  DibujaCuadro(Rec);           (*    aqui dibujo los elevadores         *)
  Rec.left := 68;              (*                                       *)
  Rec.top  := 5;               (*                                       *)
  Rec.right:= 78;              (*                                       *)
  Rec.bottom := 21;            (*                                       *)
  DibujaCuadro(Rec);           (*                                       *)
  FOR ulposx1 := 8 TO 18 DO    (*                                       *)
    GotoXY(54,ulposx1);        (*                                       *)
    FOR ulposx2:= 0 TO 8 DO    (*                                       *)
        Write(CHR(196));       (*                                       *)
    END;                       (*                                       *)
    ulposx1:=ulposx1+2 ;
  END;                         (*                                       *)
  FOR ulposx1 := 8 TO 18 DO
  GotoXY(69,ulposx1);          (*                                       *)
    FOR ulposx2 := 0 TO 8 DO   (*                                       *)
       Write(CHR(196));
    END (* for *);             (*                                       *)
    ulposx1 := ulposx1+2;
  END;                         (*****************************************)
  GotoXY(54,4);
  WriteString("Elev 11");
  GotoXY(69,4);
  WriteString("Elev 12");
  GotoXY(52,7);
  Write('5');
  GotoXY(52,10);
  Write('4');
```

```
    GotoXY(52,13);
    Write('3');
    GotoXY(52,16);
    Write('2');
    GotoXY(52,19);
    Write('1');
    GotoXY(67,7);
    Write('5');
    GotoXY(67,10);
    Write('4');
    GotoXY(67,13);
    Write('3');
    GotoXY(67,16);
    Write('2');
    GotoXY(67,19);
    Write('1');
    Rec.left := 0;
    Rec.top  := 2;
    Rec.right  := 79;
    Rec.bottom := 22;
    DibujaCuadro(Rec);
    Screen[16,58].char := CHR(219);
    Screen[16,59].char := CHR(219);
    Screen[16,73].char := CHR(219);
    Screen[16,74].char := CHR(219);
    GotoXY(1,23);WriteString("AL:");
    GotoXY(1,24);WriteString("ME:");
                                (*                                      *)
    Inicializa(Terminar);       (*                                      *)
    Inicializa(men1);           (*       inicializa senales             *)
    Inicializa(men2);           (*                                      *)
    Inicializa(men3);
                                (**************************************)
    men := ' ';
    ulposx1:=16;
    ulposx2:=16;
    SetDeviceStatus(3, TRUE);
    SetDeviceStatus(4, TRUE);
    StartDevice();
    ComienzaTimer();
    StartReading();
    CreaProceso(Maestro, 16384);
    CreaProceso(MenuPri,16384);
    CreaProceso(InterpretaMensajePC,16384);
    CreaProceso(VerificaRespPC,16384);
    LOOP
      IF CompareStr(men,"*") = 0  THEN EXIT END;
    END;
    StopReading;
    StopDevice;
    ParaTimer;
    SetDeviceStatus(6,TRUE);
    ENABLE;
END Monitor2.
```

```
                DEFINITION MODULE utilerias;
EXPORT QUALIFIED criticalSectionPtr,ConvFechaADias,
                 ConvHoraAHoras,SumaFechaHora,
                 ConvStringToHora,ConvStringToFecha,
                 DameHoraSistema,DameFechaSistema;
TYPE
     BooleanPtr = POINTER TO BOOLEAN;     -|
VAR
    criticalSectionPtr:BooleanPtr;

  PROCEDURE DameHoraSistema(VAR HORA,MINUTO,SEGUNDO:CARDINAL);

  PROCEDURE DameFechaSistema(VAR ANIO,MES,DIA:CARDINAL) ;


  PROCEDURE ConvFechaADias(dia,mes,anio:CARDINAL; VAR SumaDias:CARDINAL) ;

  PROCEDURE ConvHoraAHoras(horas,mins,segs:ARRAY OF CHAR; VAR Horas : REAL)
    (* este procedimiento convierte una hora compuesta por HH:MM:SS
       en Horas
    *)
  PROCEDURE SumaFechaHora(Dias:CARDINAL; Horas:REAL; VAR TotalHoras:REAL);
  (* Este procedimiento se encarga sumar las horas en los dias
     mas las horas dadas e indica cuantas horas han pasado
     desde 1984 a la fecha y hora dada
  *)
  PROCEDURE ConvStringToFecha(Fecha : ARRAY OF CHAR;
                              VAR DiaCard,
                                  MesCard,
                                  AnioCard:CARDINAL):BOOLEAN ;

  PROCEDURE ConvStringToHora(str:ARRAY OF CHAR;
                             VAR hh,mm,ss:CARDINAL):BOOLEAN ;
    (* Este procedimiento se encarga de convertir una hora
       dada como string a numerica y valida que este dentro de
       rangos permitidos *)

END utilerias.
```

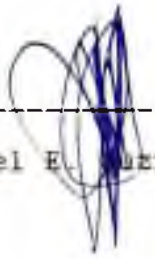
```
            IMPLEMENTATION MODULE utilerias;
(*
Title   : utilerias
Comment : Este modulo provee diversos procedimientos que son utiles
          a diversos modulos, procedimientos como DameHoraSistema,
          el cual me da la hora del sistema, ConvStringToHora, el cual
          valida una hora dada en forma de string y entrega el resultado
          en cardinal, Etc.
LastEdit: 8 de Abril de 1987
Author  : Felipe Verdalet Guzman
System  : LOGITECH MODULA-2/86
*)
  FROM SYSTEM IMPORT ADR,ADDRESS,SETREG,GETREG,SWI,AX,ES,BX,DOSCALL;
  FROM NumberConversion IMPORT StringToCard;
  FROM Strings IMPORT Length, Copy,Concat;
  FROM RealConversions IMPORT StringToReal;
  FROM Conversions IMPORT ConvertCardinal;
  TYPE
     Register =
         RECORD
         CASE BOOLEAN OF
           TRUE :
           X : CARDINAL;
         | FALSE :
           L, H : CHAR;
         END;
         END;
  VAR
    aux:ADDRESS;
    Pausa:CHAR;


    PROCEDURE DameHoraSistema(VAR HORA,MINUTO,SEGUNDO:CARDINAL) ;
       (* este procedimiento se encarga de obtener la hora del
          del sistema y regresarla en las variables HORA,MINUTO,
          SEGUNDO *)
    VAR
       horaminuto,segmiliseg:Register;
    BEGIN
        (* Este llamado al DOS obtiene la hora del sistema *)
        DOSCALL(2CH,horaminuto.X,segmiliseg.X);
        HORA    := ORD(horaminuto.H);
        MINUTO  := ORD(horaminuto.L);
        SEGUNDO := ORD(segmiliseg.H);
    END DameHoraSistema;

    PROCEDURE DameFechaSistema(VAR ANIO,MES,DIA:CARDINAL) ;
      (* esta rutina se encarga de obtener la fecha del sistema
         y la regresa en las variables ANIO,MES,DIA              *)
    VAR
      year,monthday:Register;
    BEGIN

      (* Traer la fecha del sistema                          *)
       DOSCALL(2AH,year.X,monthday.X);
       ANIO := year.X;
       MES  := ORD(monthday.H);
       DIA  := ORD(monthday.L);
    END DameFechaSistema;

PROCEDURE ConvFechaADias(dia,mes,anio:CARDINAL; VAR SumaDias:CARDINAL) ;
```

```
        anioaux         : CARDINAL;
        mesaux          : CARDINAL;
BEGIN
  (* Tomo Como Base 1984 ya que es anio bisiesto,
     La Rutina me indica cuantos dias han transcurrido
     de 1984 a la fecha                                    *)
  anioaux       := 0;
  mesaux        := 0;
  SumaDias      := 0;
  AnioBisiesto := 0;
  anio          := anio-1900;
  FOR anioaux := 84 TO anio-1 DO
    IF ((anioaux MOD 4) = 0) THEN
      AnioBisiesto := 1;
    ELSE
      AnioBisiesto := 0;
    END (* if *);
    SumaDias := SumaDias + 365 + AnioBisiesto;
  END (* for *);
  FOR mesaux := 1 TO mes-1 DO
    CASE mesaux OF
      01:                                          (* Enero *)
         SumaDias := SumaDias + 31;
    | 02:                                          (* Febrero *)
         IF ((anio MOD 4) = 0) THEN
            SumaDias := SumaDias + 29;             (* si es bisiesto *)
         ELSE
            SumaDias := SumaDias + 28;             (* si no es bisiesto *)
         END (* if *);
     | 03:                                         (* Marzo *)
          SumaDias := SumaDias + 31;
     | 04:                                         (* Abril *)
          SumaDias := SumaDias + 30;
     | 05:                                          (* Mayo *)
          SumaDias := SumaDias + 31;
     | 06:                                          (* Junio *)
          SumaDias := SumaDias + 30;
     | 07:                                          (* Julio *)
          SumaDias := SumaDias + 31;
     | 08:                                           (* Agosto *)
           SumaDias := SumaDias + 31;
     | 09:                                           (* Septiembre *)
          SumaDias := SumaDias + 30;
     | 10:                                           (* Octubre *)
          SumaDias := SumaDias + 31;
     | 11:                                           (* Noviembre *)
          SumaDias := SumaDias + 30;
     | 12:                                           (* Diciembre *)
          SumaDias := SumaDias + 31;
    END (* case *);
  END (* for *);
  SumaDias := SumaDias + dia;
END ConvFechaADias;

PROCEDURE ConvHoraAHoras(horas,mins,segs:ARRAY OF CHAR; VAR Horas : REAL) ;
  (* este procedimiento convierte una hora compuesta por HH:MM:SS
     en Horas
  *)
  VAR
      horaaux,minaux,segaux:REAL;
      Done:BOOLEAN;
BEGIN
  StringToReal(horas,horaaux,Done);       (* convierto las horas a Real    *)
  StringToReal(mins,minaux,Done);         (* convierto los minutos a Real *)
```

```
  segaux := segaux/3600.0;                (* convierto segundos a horas   *)
  minaux := minaux/60.0;                  (* convierto minutos a horas    *)
  Horas:= horaaux + minaux + segaux;      (* sumo las horas               *)
END ConvHoraAHoras;

PROCEDURE SumaFechaHora(Dias:CARDINAL; Horas:REAL; VAR TotalHoras:REAL);
(* Este procedimiento se encarga sumar las horas en los dias
   mas las horas dadas e indica cuantas horas han pasado
   desde 1984 a la fecha y hora dada
*)

   VAR
       diasaux : REAL;
       diasaux1: ARRAY [0..6] OF CHAR;
       Done:BOOLEAN;
BEGIN
   Dias := Dias * 24;                     (* convierto los dias a horas   *)
   ConvertCardinal(Dias,7,diasaux1);      (* convierto los dias a string  *)
   StringToReal(diasaux1,diasaux,Done);   (* convierto el string a Real   *)

   TotalHoras := diasaux + Horas;         (* obtengo el total de horas    *)
                                          (* transcurridos desde 1984     *)
END SumaFechaHora;

PROCEDURE ConvStringToFecha(Fecha : ARRAY OF CHAR;
                            VAR DiaCard,
                                MesCard,
                                AnioCard:CARDINAL):BOOLEAN ;
(* este procedimiento de encarga de validar que una fecha sea dada
   correctamente, checando que el anio no sea menor que el del
   sistema (es importante que la fecha del sistema sea la correcta)
   ademas checa que los meses y los dias sean correctos, tambien
   toma en cuenta los anios bisisetos,
   Entrada : Fecha en forma de string;
   Salida  : Cierto si es correcta y ademas la fecha convertida
                    a Anio, Mes, Dia.
             Falso si no fue correcta.
*)
VAR
    ANIO,MES,DIA:CARDINAL;          (* Contienen la fecha del sistema *)
    Dia,Mes:ARRAY [0..1] OF CHAR;
    Anio:ARRAY[0..3] OF CHAR;
    Done:BOOLEAN;
    meses: SET OF [1..12];
    AnioBisiesto:BOOLEAN;
    T:CARDINAL;
    Date:ARRAY[0..7] OF CHAR;
BEGIN

  AnioBisiesto := FALSE;
  IF Length(Fecha) <> 8 THEN
    RETURN(FALSE);
  ELSE
    Copy(Fecha,0,2,Dia);
    Copy(Fecha,3,2,Mes);
    Copy(Fecha,6,2,Anio);
    Concat("19",Anio,Anio);
    Concat(Dia,Mes,Date);
    Concat(Date,Anio,Date);
    FOR T:=0 TO (Length(Date)-1) DO
      IF (Date[T] < '0') OR (Date[T] > '9') THEN RETURN(FALSE); END;
    END ; (* for *)
```

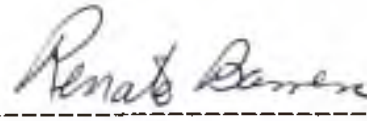
```
      DameFechaSistema(ANIO,MES,DIA);
      IF (AnioCard MOD 4) = 0 THEN AnioBisiesto := TRUE; END;
      IF  ( AnioCard >= ANIO) THEN
        IF (MesCard >0) AND (MesCard <13) THEN
              IF (DiaCard >0) AND (DiaCard<32)  THEN
                IF MesCard = 2 THEN
                  IF  NOT(AnioBisiesto) AND (DiaCard = 29)THEN
                    RETURN(FALSE);
                  END;(* if not bisiesto *)
                END; (* if febrero *)
                RETURN(TRUE);
              ELSE
                RETURN(FALSE);
              END; (* if dia en dias *)
        ELSE
          RETURN(FALSE);
      END (* if mes en meses *);
    ELSE
      RETURN(FALSE);
    END (* if anio > anioactual *);
  END (* if fecha <> 8 *);
  END ConvStringToFecha;

  PROCEDURE ConvStringToHora(str:ARRAY OF CHAR;
                             VAR hh,mm,ss:CARDINAL):BOOLEAN ;
    (* Este procedimiento se encarga de convertir una hora
       dada como string a numerica y valida que este dentro de
       rangos permitidos *)
    VAR
        hh1,mm1,ss1:ARRAY[0..1] OF CHAR;
        Done : BOOLEAN;
        tiempo:ARRAY[0..5] OF CHAR;
        t:CARDINAL;
  BEGIN
      IF Length(str) <> 8 THEN RETURN(FALSE); END;

      Copy(str,0,2,hh1);
      Copy(str,3,2,mm1);
      Copy(str,6,2,ss1);
      Concat(hh1,mm1,tiempo);
      Concat(tiempo,ss1,tiempo);
      FOR t:=0 TO (Length(tiempo)-1) DO
        IF (tiempo[t] < '0') OR (tiempo[t] > '9') THEN RETURN(FALSE); END;
      END ; (* for *)
      StringToCard(hh1,hh,Done);
      StringToCard(mm1,mm,Done);
      StringToCard(ss1,ss,Done);
      IF (hh >= 0) AND (hh <24) AND (mm >= 0) AND (mm <60) AND
         (ss >=0) AND (ss<60) THEN
         RETURN(TRUE);
      ELSE
         RETURN(FALSE);
      END; (* if *)
  END ConvStringToHora;

BEGIN

  GETREG(AX, 3400H);
  SWI(21H);
  GETREG(ES, aux.SEGMENT);
  GETREG(BX, aux.OFFSET);
  CriticalSectionPtr := BooleanPtr(aux);
  (* utilerias.
```

El jurado designado por la sección de Computación del Departamento de Ingeniería Eléctrica del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó esta tesis el 23 de Septiembre de 1987.



Dr. Manuel E. Guzmán Rentería

Dr. Armando Maldonado Talamantes



Dr. Renato Barrera Rivera

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL
INSTITUTO POLITECNICO NACIONAL

BIBLIOTECA DE INGENIERIA ELECTRICA
FECHA DE DEVOLUCION

El lector está obligado a devolver este libro antes del vencimiento de préstamo señalado por el último sello.

30 SET. 1988

DEVOLUCION

AUTOR VERDALET GUZMAN, F.

TITULO SISTEMA SUPERVISOR DE PROCESOS EN UNA COMPUTADORA PERS.

CLASIF. XM 87.19 RGTRO. BI 10,813

NOMBRE DEL LECTOR	FECHA PREST.	FECHA DEVOL.
Ma. Teresa Virgen	[illegible]	4/3/94
Dr. Horacio L.	3-I-94	31/1/94