CM

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

INTERFAZ GRAFICA PARA LA MAQUINA MONOPUTER 2

Tesis que presenta el Ing. Eduardo Cortés Gil para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERIA ELECTRICA con opción en COMPUTACION.

Trabajo dirigido por el Dr. Jan Janecek Hyan.

México D.F., Junio de 1990

A MIS PADRES Y HERMANOS CON PROFUNDO RESPETO Y CARIÑO.

AGRADECIMIENTOS

Antes que nada quiero expresar mi agradecimiento al Dr. Jan Janecek por sus observaciones y sugerencias en el presente trabajo.

Al CINVESTAV-SECCION DE COMPUTACION por la ayuda brindada. Y al Sistema Nacional de Educación Tecnológica (COSNET), por la ayuda económica que me proporcionó.

INDICE

INTRODUCCION

Muchas tareas, como la generación de gráficas, la simulación de procesos etc. son actividades con una gran avidez de potencia de cálculo, de procesadores más potentes. Y esta necesidad de cálculo es cada vez mayor en las nuevas aplicaciones. La máquina monoputer2 debido a sus caracteristicas, explicadas más adelante, nos presenta una forma de satisfacer esta necesidad. Sin embargo esta no permite el manejo de gráficas, restringiendose unicamente a la manipulación de texto. Así el presente trabajo ofrece una solución al proporcionar la herramienta necesaria para realizar gráficas mediante esta máquina, empleando para ello una computadora AT como nodriza.

Primero se da una breve descripción de lo que es un transputer y que es la máquina monoputer 2; y se describe el sistema en el que se encuentra la monoputer, asi como la forma de comunicación que se tiene en cuanto al menejo de texto; todo esto en el primer capítulo.

En el segundo capítulo se presenta el centro de la interfaz gráfica la cual consiste de una serie de programas que permiten el manejo tanto de texto como de gráficas . Aqui se describe la forma en que se realiza la comunicación entre la monoputer y la nodriza para realizar la generación y el despliegue de gráficas. Esta parte es muy importante ya que deja abierto totalmente el camino para realizar posteriores cambios si es que se desea.

En el tercer capítulo se describe una forma diferente de trabajo, ya que permite a cualquier programa de usuario que corra en la nodriza, accesar a la monoputer. Esta parte consiste

de varios programas que permiten ver a la monoputer como un recurso más de la máquina nodriza.

El cuarto capítulo muestra una mejora importante. Se trata de un servidor residente el cual esta formado por varios programas que permitiran el mejor empleo de la máquina nodriza cuando se corren programas en la monoputer que consumen mucho tiempo, ya que la nodriza estara libre en tanto la monoputer no la accese.

Al final del trabajo, en el quinto capítulo, se muestran algunos ejemplos de aplicación, como comparación en cuanto a la velocidad para realizar gráficas, entre la monoputer y otras máquinas.

Todos los programas fuentes de este trabajo, se encuentran en la sección de computación del departamento de Ingeniería Eléctrica en el Centro de Investigación y de Estudios Avanzados del Politécnico.

Eduardo Cortés Gil.

CAPITULO 1
LA RELACION MONOPUTER-NODRIZA

1.1.- EL TRANSPUTER Y LA MAQUINA MONOPUTER 2.

La necesidad cada vez mayor de cálculo exige la existencia de computadoras más potentes. Sin embargo la arquitectura de las supercomputadoras actuales no les permite extenderse mucho más alla de sus actuales alcances. Por lo que se han estado haciendo investigaciones para crear otro tipo de computadoras, los llamados procesadores paralelos. Así en 1979, un grupo de investigadores que trabajaba para la empresa británica Inmos comienza los trabajos sobre un nuevo componente, destinado a trabajar en paralelo con muchos otros de su tipo, y de esta manera en 1985 aparece el primer Transputer. Luego, el Transputer se ha convertido en el nombre genérico de una nueva generación de componentes usados como elementos independientes pero capaces de ser integrados en grandes redes para trabajar en forma paralela.

Existen varios tipos de transputers sin enbargo todos ellos cuentan con el mismo conjunto básico de instrucciones. El T414 [6], el cual cuenta con un procesador capaz de manejar datos de 32 bits; con una memoria interna de 2Kb; con una interfaz muy veloz la cual permite accesar una memoria externa; con dos timers empleados para realizar los rapidos cambios de contexto; y cuatro enlaces de comunicación independientes que le permiten intercambiar datos a una velocodad máxima de 10 Mbits/s.

El último miembro de los Transputers hasta ahora es el T800 [7],[8],(Fig. 1); del cual existen algunas variantes, cuyos enlaces de comunicación tienen una velocidad máxima de 20 Mbits/s; cuenta además con una unidad de punto flotante; y cuenta con 4 Kbytes de memoria interna en lugar de los 2 Kbytes

del T414. El T800 tiene por si solo un conjunto de ventajas superiores a las de la mayoria de los procesadores comerciales, ya que permite que sus seis unidades fundamentales

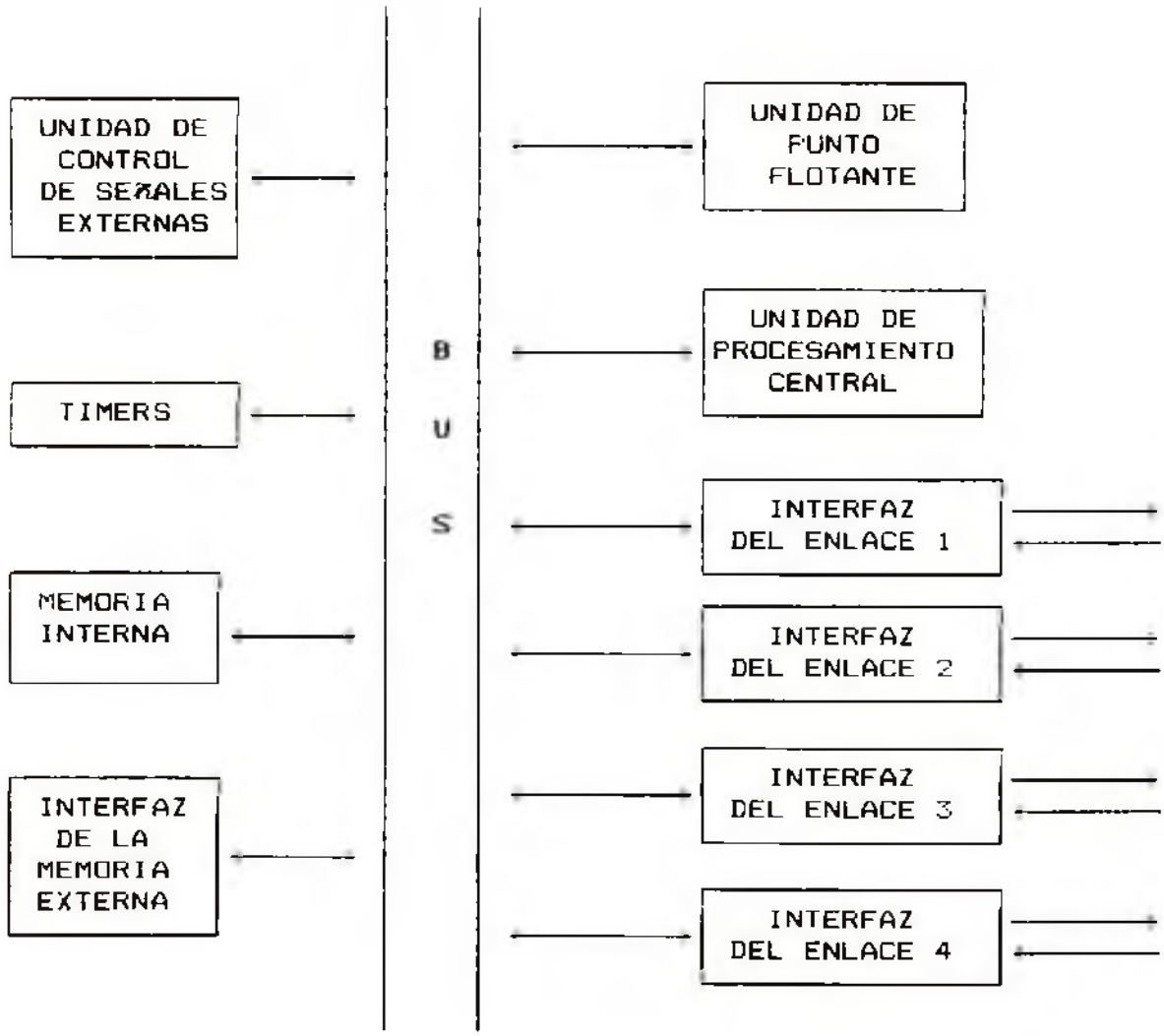


Fig. 1. Diagrama general del Transputer T800

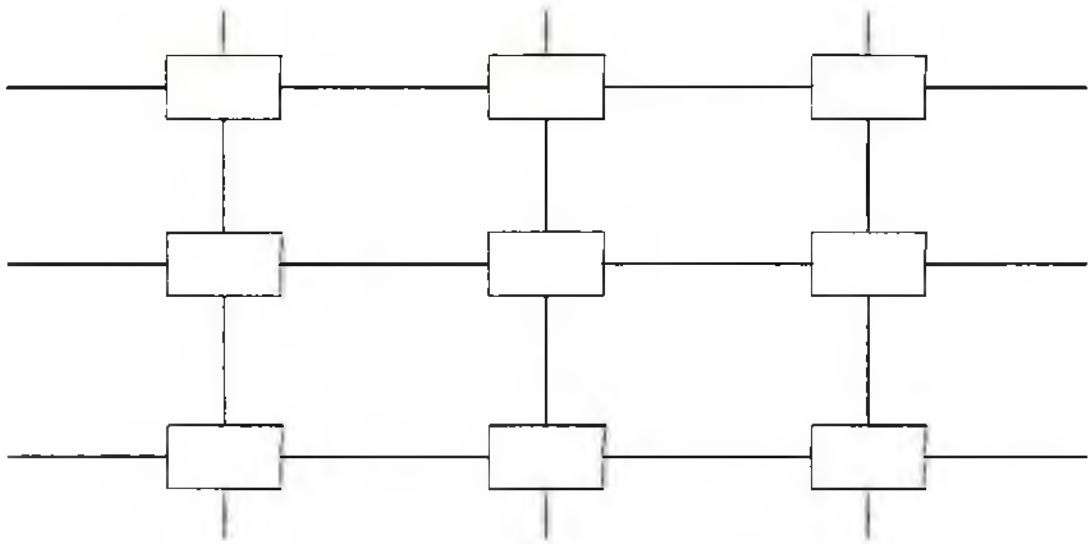
(CPU, FPU y sus cuatros enlaces) trabajen simultáneamente, pero esta ventaja se nota aún mucho más cuando se conecta a una red de varios transputers, para que trabajen en forma paralela.

Los transputers proporcionan diferentes formas y grados de paralelismo. Permiten un autentico procesamiento paralelo empleando un sistema de varios transputers interconectados, como en los ejemplos de la figura2. También permiten un cuasi-paralelismo, es decir la ejecución de más de un proceso en un mismo transputer. Esto se logra gracias al administrador de procesos microprogramado con que se se cuenta, el cual realiza los cambios de contexto empleando tan solo uno cuantos micro-segundos, dando como resultado un rapido cuasi-paralelismo. Asi tanto el grado de paralelismo como el poder de procesamiento se incrementa en proporción al número de transputers empleados en un sietema.

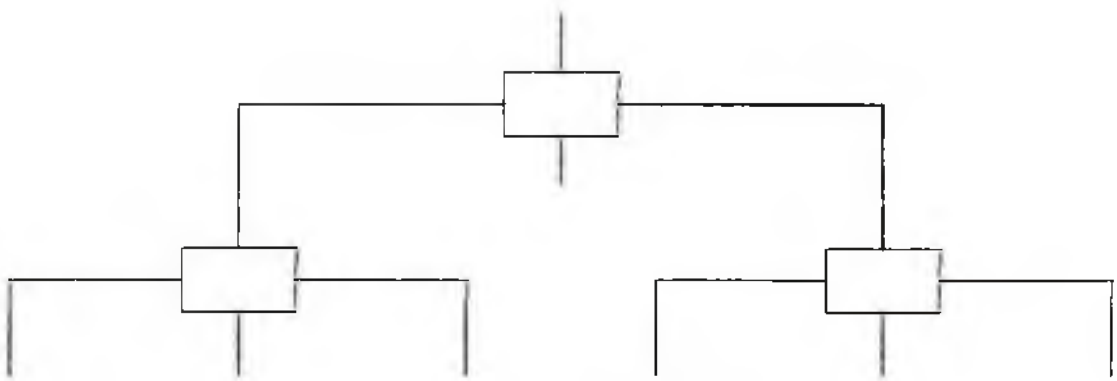
Para trabajar con los transputers se emplea un lenguajeparalelo de alto novel, llamado Occam [1],[2],[9]; el cual permite explotar las características de esta máquina. La comunicación entre procesos parelelos, es llevada a cabo mediante el envío de mensages usando canales unidireccionales explicitamente definidos. Hay que hecer notar que esta comunicación es sincrona.

Generalmente los transputers trabajan bajo el control de una máquina nodriza. En nuestro caso, un T800, se encuentra implantada dentro de una máquina AT. Este transputer se encuentra formando parte de la computadora llamada monoputer 2 la cual cuenta con 2 Mbytes de memoria externa y se encuentra en una tarjeta, como expanción para la AT que usa como nodriza.

Estructura lineal



Estructura de matriz



Estrucrura de árbol

Fig. 2. Estructuras de sistemas con varios Transputers

1.2.- LA COMUNICACION MONOPUTER-NODRIZA.

Como ya se mencionó en la introducción, la máquina monoputer 2 se encuentra implantada dentro de una máquina AT, la cual le sirve como nodriza. La monoputer se comunica con esta nodriza através de un conjunto de puertos de ésta última. De esta forma la nodriza ve a la monoputer simplemente como un conjunto de puertos desde los cuales recibe o envía información. Por otro lado la monoputer ve la nodriza como un par de canales de comunicación, uno para enviar y el otro para recibir información. Esto se muestra en el dibujo de la figura 3.

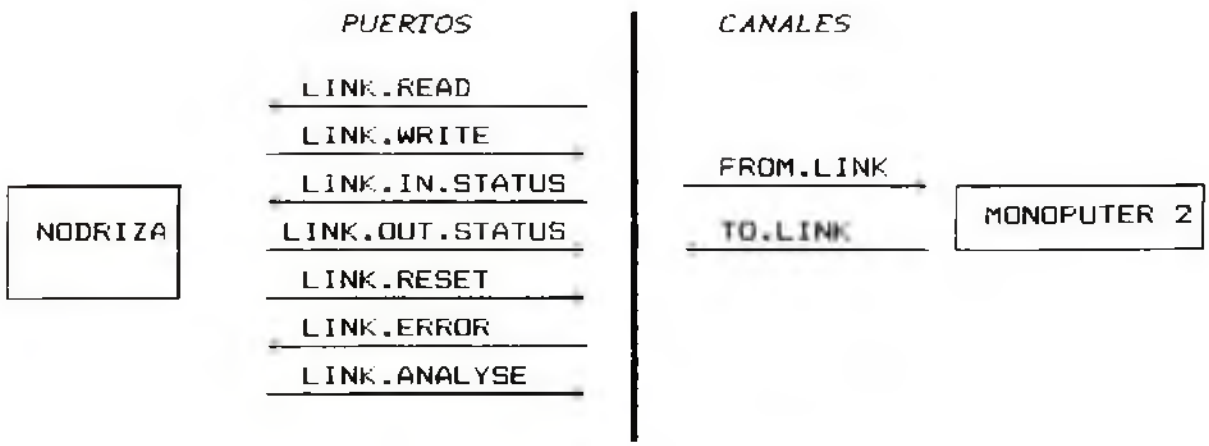


Fig. 3. Enlace entre la momoputer y la nodriza.

En seguida daremos la descripción de cada uno de los puertos y canales empleados para la comunicación.

LOS PUERTOS

Estos corresponden a varios puertos físicos de la máquina nodriza, y son el medio de comunicación de ésta con la monoputer. Enseguida describiremos para que se emplea cada uno de estos puertos:

LINK.READ.- Es por este puerto que la nodriza recibe los datos que le envía la monoputer.

LINK.WRITE.- Através de este puerto la máquina nodriza le envía datos a la monoputer.

LINK.IN.STATUS .- Este puerto es empleado por la monoputer para indicar a la nodriza que espera le envie algun mensaje. Además es empleado para habilitar operaciones de acceso directo a memoria.

LINK.OUT.STATUS.- Este puerto es empleado por la monoputer para indicar a la nodriza que tiene algun dato para ella. También es usado para habilitar operaciones de acceso directo a memoria.

LINK.RESET.- Este puerto es empleado por la máquina nodriza para inicializar a la monoputer.

LINK.ERROR.- Por medio de este puerto se indica si ha sucedido algun error en el sistema de la monoputer.

LINK.ANALYZE.- Este puerto es empleado junto con LINK.RESET para inicializar la monoputer pero preservando el estado del procesador de esta máquina.

LOS CANALES

Estos son canales lógicos de la monoputer asociados con canales físicos de esta máquina, los cuales emplea para comunicarse con la máquina nodriza. A continuación describiremos como el uso que se le da a estos canales:

FROM.LINK.- Este canal es empleado por la monoputer para recibir todos los datos que envie la máquina nodriza.

TO.LINK.- Este canal lo emplea la monoputer para enviar mensajes hacia la nodriza.

Para la comunicación entre las dos máquinas se cuenta con varios programas que se encargan de definir y manejar los protocolos necesarios. Existe un programa servidor que corre en la nodriza, el cual se encarga de recibir e interpretar la información que le envia la monoputer y realizar las operaciones que se deban ejecutar en la máquina donde corre. Este programa es llamado afserver [3], y fué adquirido junto con la monoputer en su forma ejecutable. En la interfaz gráfica este programa es llamado MyAFServ.

Por otro lado, debe ejecutarse en la transputer un programa llamado harness [3], para el servidor anterior y MyHarn para la interfaz gráfica; el cual se encarga de llamar a la rutina del usuario y de crear un proceso llamado filter [3], para el servidor anterior y MyFilt para la interfaz gráfica; el cual asu vez crea dos procesos que trabajan en forma paralela al proceso harness. La función del programa filter es entablar la comunicación con la nodriza del lado de la monoputer, esto lo podemos ver en forma esquemática en el dibujo de la figura 4.

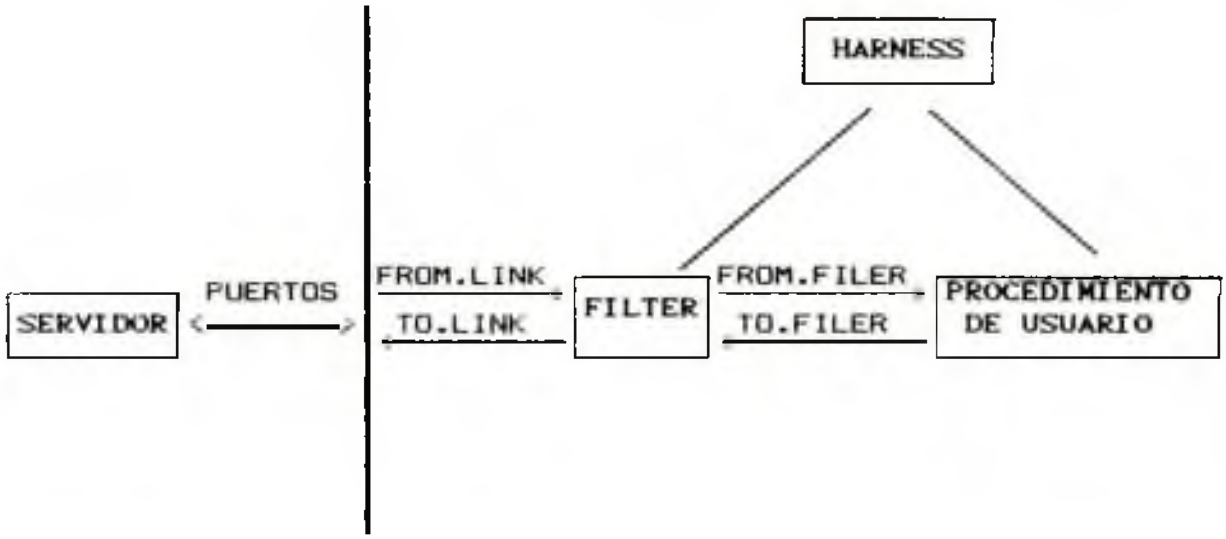


Fig. 4. Procesos de enlace entre monoputer y nodriza

Como vemos, existen dos programas que se encargan de la comunicación entre la monoputer y la nodriza que son: el servidor y el filter. Este último define el formato del protocolo para el envio y recepción de información entre la monoputer y la máquina nodriza.

El sistema de comunicación está dividido en tres niveles. El primero de ellos, el más bajo, esta basado en un protocolo básico, el cual es manejado directamente por los procesos filter y AFServer en el antiguo servidor y por los procesos myfilt y MYAFServ en la interfaz gráfica. El segundo nivel se basa en un conjunto de primitivas que sirven para definir el protocolo del

tercer nivel. Ahora pasaremos a explicar cada un de estos niveles.

1.2.1.-EL PRIMER NIVEL DE COMUNICACION

El primer nivel de comunicación está fundamentado en un protocolco básico etiquetado.

El formato general del protocolo básico, mostrado en la figura 5, consiste de un entero, usado como etiqueta para definir el tipo, seguido del dato o bloque de datos.

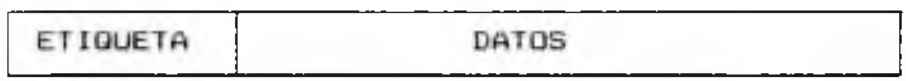


Fig. 5.-Formato de protocolo para la transferencia de datos.

El protocolo básico de comunicación emplea dos canales: 1). from.filer el cual es usado para enviar datos del proceso 'filter' al programa del usuario, y 2). to.filer, el cual es empleado para enviar datos desde el programa del usuario al proceso 'filter'. Las datos son enviados y recibidos por medio de un protocolo etiquetado, en el cual la etiqueta indica el tipo

de datos que sigue inmediatamente a ésta. Las etiquetas son las siguientes:

```
bool.value       IS BYTE 0:     BOOL
byte.value       IS BYTE 1:     BYTE
int.value        IS BYTE 2:     INT
int16.value      IS BYTE 3:     INT16
int32.value      IS BYTE 4:     INT32
int64.value      IS BYTE 5:     INT32
real32.value     IS BYTE 6:     REAL32
real64.value     IS BYTE 7:     REAL64
nilrecord.value IS BYTE 8:      [BYTE]BYTE
record8.value    IS BYTE 9:     [BYTE]BYTE
record.value     IS BYTE 10:    [INT]BYTE
record16.value   IS BYTE 11:    [INT16]BYTE
record32.value   IS BYTE 12:    [INT32]BYTE
```

1.2.2.- EL SEGUNDO NIVEL DE COMUNICACION

Los tipos de datos usados al accesar el servidor son enteros o registros, los cuales son enviados y recibidos como mensajes. Para lo cual se cuenta con el conjunto de primitivas descritas acontinuación:

read.integer.- Se emplea para la transmisión de un entero de 32 bits de la máquina nodriza hacia la monoputer.

write.integer.- Se usa para el envío de un entero de 32 bits desde la monoputer a la nodriza.

read.record.- Empleada para el envio de una conjunto de bytes de la nodriza a la monoputer.

write.record.- Usada para la trasmisión de un arreglo de bytes de la monoputer a la nodriza.

1.2.3.- EL TERCER NIVEL DE COMUNICACION

Para que un programa de usuario que corra en la monoputer mantenga comunicación con la nodriza debe hacerlo empleando las primitivas del segundo nivel o bien mediante el uso de un conjunto de rutinas de alto nivel. Estas rutinas están definidas en la librería flibs [3] en el antiguo servidor y en la librería Myflibs para la interfaz gráfica.

El acceso al servidor es logrado empleando el protocolo en el cual el programa de usuario siempre debe enviar primero un entero el cual corresponde al código de la función que el servidor debe ejecutar, luego enviar una serie de datos o bloques de datoscomo parametros, para luego recibir también una serie datos o bloques de datos como respuesta, para finalmente recibir un entero con el resultado de la operación.

Existen algunos puntos que hay que tomar en cuenta al accesar al Servidor ademas de este protocolo:

- Los archivos y streams pueden existir en una de dos formas:
  Binaria (AccesMethod IS 1:), en al cual el archivo o el stream debe consistir de bytes representando algun tipo de dato.

Texto (AccesMethod=1:), en el cual el filer inserta retornos de carro como sea necesario para dar a este archivo el formato de un archivo de texto de DOS [4].

- Los archivos son accesados por sus nombres dados por DOS, y pueden ser abiertos para lectura, escritura, o actualización.
- Un número máximo de ocho archivos pueden estar abiertos al mismo tiempo, si bien esto puede restringirse más de acuerdo a la configuración de DOS.

- Las operaciones que involucran archivos o streams sólo pueden mover un máximo de 512 bytes en un tiempo, para enviar bloques de mayor tamaño debe enviarse como varios bloques menores, realizando la operación tantas veces como sea necesario.

-Se cuenta con un conjunto estandar de valores constantes:

Método de acceso	
BinaryByteStream	AccesIS 0:
TextByteStream	AccessIS 1:
Modo de apertura	
Read.Mode	IS 0:
Write.Mode	IS 1:
Update.Mode	IS 2:
Modo de existencia	
Old.File	IS 0:
New.File	IS 1:
Conecciones	
Screen.Use	IS 0:
Keyboard.Use	IS 1:
File.Use	IS 2:
Temp.Use	IS 3:
Parameter.Use	IS 4:

Opciones para cerrar

Close.Option	IS 0:
CloseDel.Option	IS 1:

Los cuales pueden definirse para su uso en las rutinas que accesan al servidor.

- Los valores que corresponden a cada una de las funciones descritas son los siguientes:

AlienTerminate.Cmd	0	OpenInputStream.Cmd	3
RunCommand.Cmd	126	OpenOutputStream.Cmd	4
Get.Cur.Cmd	15	StreamAcces.Cmd	5
Set.Cur.Cmd	16	StreamStatus.Cmd	6
ReceiveBlock.Cmd	30	StreamFile.Cmd	7
SendBlock.Cmd	31	StreamConnect.Cmd	23
Terminate.Cmd	24	CloseStream.Cmd	11
RenameFile.Cmd	127	ReadBlock.Cmd	12
OpenFile.Cmd	1	WriteBlock.Cmd	13
OpenTemp.Cmd	2	WriteBlockAttr.Cmd	14
		Scr.Clear.Cmd	17

Enseguida daremos la descripción de las secuencias que definen el protocolo del tercer nivel las cuales son realizadas por las rutinas de este nivel de comunicación para accesar las diferentes funciones del servidor. Todas las funciones accesadas mediante estas secuencias son para el intercambio de información en forma de texto, y si bien algunas de ellas existen en el servidor anterior, éstas son parte de la nueva interfaz, las cuales aunque para el usuario funcionen igual no necesariamente trabajan de la misma manera.

Poner un resultado

Esta función asigna un valor, el cual es regresado por el servidor al sistema operativo cuando este termina. La secuencia para su llamado es:

```
write.integer(SetResult.Cmd)
write.integer(valor)
read.integer(resultado)
```

Correr un comando

Esta función ejecuta un comando dado en la máquina nodriza. La secuencia para llamarla es:

```
write.integer(RunCommand.Cmd)
write.record(lineadecomando)
read.integer(resultado)
```

Recibir un bloque

Esta función sirve para leer un bloque de memoria de la máquina nodriza y copiarla a un bloque de memoria de la transputer.
La direccion inicial de bloque en la nodriza (direccion.fuente) es un entero de 32 bits. Los 16 bits más significativos dan el segmento de memoria y los 16 bits menos significativos dan el offset dentro del segmento. Su secuencia de llamado es como sigue:

```
write.integer(ReceiveBlock.cmd)
write.integer(direccion.fuente)
write.integer(bloque.tamano)
read.record(bloque.tamano,bloque)
read.integer(resultado)
```

Enviar un bloque

Esta función envia un bloque de datos de la memoria de la monoputer a la memoria de la nodriza en la dirección dada por direccion.destino. Su secuencia de llamado es como sigue:

```
write.ineteger(SendBlock.Cmd)
write.integer(direccion.destino)
write.record(bloque)
read.integer(bloque.tamano)
read.integer(resultado)
```

Terminar

Esta función termina el proceso filer, pero antes de hacerlo debe haberse cerrado todos los archivos y streams que se hallan abierto. Su secuencia de llamado es la siguiente:

```
write.integer(Terminate.Cmd)
read.integer(resultado)
```

Abrir un archivo

Esta función abre un archivo en la máquina nodriza, empleando el modo de acceso especificado y el tamaño del registro y regresa un identificador para dicho archivo en stream.id. Su secuencia de llamado es la siguiente:

```
write.integer(OpenFile.Cmd)
write.record(nombre.archivo)
write.integer(acceso.metodo)
write.integer(abrir.modo)
write.integer(existencia.modo)
write.integer(longitud.registro)
read.integer(stream.id)
```

```
read.integer(resultado)
```

Esta tiene los siguientes códigos especiales de error:

1	Nombre de archivo demasiado largo
2	Modo de acceso no permitido
3	Modo de apertura no permitida
4	Modo de existencia no permitida
5	Longitud de registro no valida

Abrir un archivo en forma temporal

Esta función crea y abre un archivo temporal, regresando un identificador para él. Dicho archivo es borrado al momento de cerrarse. La secuencia para llamar a esta función es como sigue:

```
write.integer(OpenTemp.Cmd)
write.integer(acceso.metodo)
write.integer(longitud.registro)
read.integer(stream.id)
read.integer(resultado)
```

Renombrar un archivo

Esta función sirve para cambiar el nombre a un archivo existente en la máquina nodriza.

Su secuencia de llamado es la siguiente:

```
write.integer(RenameFile.Cmd)
write.record(nombre.viejo)
write.record(nombre.nuevo)
read.integer(resultado)
```

código especial de error

1 Operación fallida

Abrir un stream para entrada

Esta función abre uno de los streams estandar de entrada (0 es el teclado, 1 es la linea de comandos) para sólo lectura en el modo de acceso de texto. Su secuencia de llamado es la siguiente:

```
write.integer(OpenInputStream.Cmd)
write.integer(Estd,stream.id)
read.integer(stream.id)
read.ineteger(resultado)
```

código especial de error

1 número de stream de entrada inválido

Abrir un stream para salida

Esta función abre uno de los streams estandar de salida (0 ó 1, ambos para la pantalla) para sólo escritura en modo de acceso texto. La secuencia de llamado es la siguiente:

```
write.integer(OpenOutputStream.Cmd)
write.integer(Estd.stream.no)
read.integer(stream.id)
read.integer(resultado)
```

código especiale de error

1 número de stream de salida inválido

Obtener el tipo de acceso a un stream

Esta función regresa el tipo de acceso empleado al abrir un stream dado. Su secuencia de llamado es la siguiente:

```
write.integer(StreamAccess.Cmd)
write.integer(stream.id)
read.integer(acceso.metodo)
read.integer(resultado)
```

código especial de error

1	Identificador de stream inválido.

Obtener el nombre de un stream

Esta función regresa el nombre completo incluyendo subdirectorios de un stream dado. Su secuencia de llamado es como sigue:

```
write.integer(StreamFile.Cmd)
write.integer(stream.id)
read.block(longitud,nombre.archivo)
read.integer(resultado)
```

código especial de error

1	identificador de stream inválido

Cerrar un stream

Cierra un stream dado (o archivo el cual es accesado como stream). Si se emplea la opción CloseDel para cerrar el stream al archivo asociado con él es borrado, en cambio si se usa la opción Close, el nombre del archivo es adicionado al directorio. Su secuencia de llamado es como sigue:

```
write.integer(CloseStream.Cmd)
write.integer(stream.id)
write.integer(close.opcion)
read.integer(resultado)
```

código especial de error

1 Identificador de stram invalido

2 Opción de cerrar invalida

Leer un bloque

Esta función lee desde la actual posición en un stream hasta la longitud de datos a ser leída, moviendo la posición del stream más allá del último byte leído. Si se intenta leer más allá del último byte, entonces la condición de fin de archivo se hace verdadera. Su secuencia de llamado es:

```
write.integer(ReadBlock.Cmd)
write.integer(stream.id)
write.integer(registro.longitud)
read.record(bytes.leidos,registro)
read.integer(resultado)
```

código especial de error

1 Identificador de stream inválido

2 Longitud de registro no válida

Escribir un bloque

Esta función escribe un bloque dado desde la actual posición en un stream, moviendo la posición del stream más allá del último byte escrito. Su secuencia de llamado es como sigue:

```
        write.integer(WriteBlock.Cmd)
        write.integer(stream.id)
        write.record(registro)
        read.integer(registro.longitud)
read.integer(resultado)
```

código especiale de error

1 Identificador de stream inválido

2 Lonfitud de registro no válida

Obtener la posición del cursor

Esta función se emplea para obtener las coordenas del cursor en el momento que se le invoca. Hay que hacer notar que esta función no existe en el servidor anterior. Su secuencia de llamado es la siguiente:

```
        write.integer(Get.Cur.Cmd)
        read.integer(posicion.x)
        read.integer(posicion.y)
        read.integer(result)
```

Colocar el cursor en una posición dada

Esta función nos permite ubicar el cursor en la posición que se dese, dentro de la pantalla de la nodriza. Esta función tampoco existe en el servidor anterior. Su secuencia es como sigue:

```
        write.integer(Set.Cur.Cmd)
        write.integer(posicion.x)
        write.integer(posicion.y)
        read.integer(result)
```

Borra la pantalla en modo texto.

Esta función borra la pantalla de la nodriza, cuando está en modo de texto.También es una de las funciones nuevas, no existentes en el servidor anterior. Su secuencia es la siguiente:

```
write.integer(Scr.Clear.Cmd)
read.integer(result)
```

Escribir una cadena de caracteres en pantalla, dando atributo y posición.

Esta función sirve para escribir letreros en alguna posición determinada en pantalla y con un atributo dado. Su secuencia es como sigue:

```
write.integer(WriteBlockAttr.Cmd)
write.integer(atrib)
write.integer(column)
write.integer(row)
write.record(record)
read.integer(len)
read.integer(result)
```

Para todas las funciones en que no se especifica código especial de error, un cero significa que no existió error y algún valor diferente de cero significa que existe algun error.

CAPITULO 2
EL CORAZON DE LA INTERFAZ GRAFICA

Hasta ahora sólo se ha tratado la forma en que se transmite texto de una máquina a otra. En esta parte de la tesis se explica la forma en que la interfaz maneja la información sobre las gráficas.

Como ya se explicó brevemente en la introducción, esta parte consiste de un conjunto de programas orientados a mantener la comunicación entre la máquina nodriza y la monoputer realizando las tereas básicas que realizan los programas con los que se adquirió la monoputer. Sin embargo, además de estas tareas esta primera parte permite la generación y el despliegue de gráficas, así como el intercambio de la información sobre éstas, entre la monoputer y la nodriza. Este sistema es independiente del adquirido con la monoputer y totalmente nuevo, pero está hecho para que los programas escritos para trabajar con el servidor adquirido con la monputer, sean compatibles con él.

La interfaz cuenta con dos formas para el despliegue de las gráficas. En la primera forma una vez calculada cada primitiva de salida, ésta es enviada inmediatamente a la nodriza para su despliegue. En la otra forma, después de calcular la primitiva de salida, ésta en lugar de ser enviada a la nodriza se pasa a una área de memoria en la monoputer, destinada para almacenar la información de una pantalla de video , la cual puede ser enviada a la nodriza para su despliegue en pantalla, o puede ser llenada con la información proveniente de la nodriza correspondiente a su area de memoria de video.

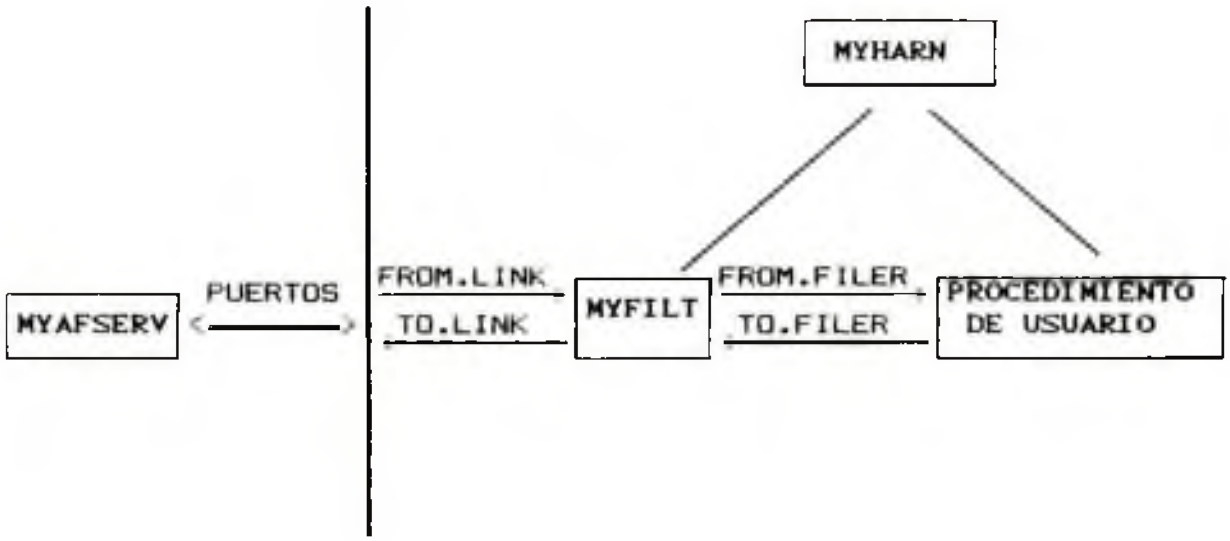


Fig. 6. El nuevo conjunto de programas para la comunicación.

En la figura 6, se muestra esquemáticamente el funcionamiento de la interfaz. Como se puede ver, se trató de apegarse lo más posible a la forma original que se tenia para manejar únicamente texto.

2.1.- EL NUEVO PROTOCOLO BASICO DE COMUNICACION Y LA INTERFAZ DEL LADO DE LA MONOPUTER

Para la realización de esta primera parte se creó un nuevo programa llamado myfilt, a partir del programa filter anterior. El nuevo programa, el cual sustituye al filter, y al igual que éste permite el envio de datos desde y hacia la monoputer empleando para ello el protocolo básico etiquetado del que ya hablamos. El nuevo programa myfilt es el proceso que trata directamente con el protocolo básico y permite el envio de los bloques de datos con la información sobre la gráficas. Dichas gráficas son enviadas en bloques de bytes y en bloques de enteros de 16 bits. De esta manera se han definido las siguientes etiquetas para el protocolo:

```
record16.value      IS  BYTE 16   []INT16
record34.value      IS  BYTE 14   []BYTE
```

La primera etiqueta es empleada para el envio y recepción de bloques de hasta 1024 enteros de 16 bits cada uno, los cuales tienen la información acerca de las coordenadas del eje de las ordenadas como de las abcisas de las primitivas de salida. Si se llegara a necesitar el manejo y envio de primitivas con más de estos 1024 puntos, se puede lograr un buen resultado dividiendo dicha primitiva en varias partes como una especie de subprimitivas, cada una con un máximo de 1024 puntos; y enviarse cada parte en forma separada como si se tratara de varias primitivas independientes. O bien modificando a MyFilt para que este nos permita manejar primitivas de un tamaño mayor, lo que resulta ser una tarea muy simple.

La segunda etiqueta es empleada para el envio y recepción de bloques, los cuales tienen la información de una pantalla

completa. Para esto se divide el bloque que representa a la pantalla en tantos sub-bloques de hasta 1024 bytes cada uno como renglones tenga la pantalla. De esta manera si se llegara a necesitar podemos tratar con resoluciones de un máximo 1024 pixeles por renglón y tantos renglones como 2^{32}, con la posibilidad de poder manejar un máximo de hasta 256 colores diferentes. Esto es suficiente para la mayoría de las aplicaciones considerando además que la nodriza es una máquina AT. Sin embargo si se llegara a necesitar más resolución o manejar un mayor número de colores sólo serían necesarios unos pequeños cambios o agregar algunas rutinas más. Para manejar una mayor resolución, es suficiente con hacer un pequeño cambio en el proceso MyFilt, para que permita manejar un mayor número de pixeles por renglón, ya que la resolución en cuanto al número de pixeles en sentido vertical que se tiene, ya es bastante grande como para necesitarse una mayor. En cuanto al número máximo de colores que se puede manejar este se puede incrementar creando las rutinas que manejen blóques de datos más grandes, las cuales serían idénticas a las que se tienen pero para manejar enteros de 16 bits lo cual nos permitiría manejar hasta 64 K colores lo cual considero sería suficiente para la mayoría de las aplicaciones; o incluso se puede hacer lo mismo pero para enteros de 32 bits.

2.2.- EL SEGUNDO Y TERCER NIVELES DE COMUNICACION EN LA INTERFAZ GRAFICA

El segundo nivel.

En seguida veremos el segundo y tercer niveles de comunicación para realizar gráficas. Básicamente son iguales que como los descritos en el capítulo anterior, pero tienen algunas partes hechas especialmente para la trasmisión de datos referidas a la primitivas de graficación y al envio de pantallas.

Para el envió de información sobre las gráficas que deben desplegarse, se necesita la trasmisión de datos y bloques de datos diferentes a los que se vieron en el capitulo anterior, para el manejo de texto. Enseguida describiremos el conjunto de primitivas especialmente realizadas para el manejo de este tipo de datos y bloques de datos, las cuales forman el segundo nivel de comunicación y están basadas en el protocolo básico descrito en la sección anterior.

write.record16.- Esta función envia un bloque de enteros de 16 bits de la monoputer a la nodriza, el cual contiene las coordenadas de alguna primitiva gráfica de salida.

read.screen.rec.- Esta función recibe un arreglo de bytes de la nodriza. Esta función es usada por readscr para leer al área de memoria correspondiente a la memoria de video en la nodriza.

write.screen.rec.- Esta función envía un arreglo de bytes a la nodriza. Esta función es empleada por writescr para escribir al area de memoria correspondiente a la memoria de video.

El tercer nivel.

Para accesar a la máquina nodriza, cuando se quiere realizar gráficas, se hace mediante las funciones de alto nivel definidas en la nueva libreria MyFlib, la cual contiene las funciones básicas de la libreria Flibs, además de un conjunto de funciones nechas para el manejo de gráficas, las cuales se basan en las primitivas del segundo nivel descritas con anterioridad. Enseguida describimos estas funciones, las cuales pertenecen al tercer nivel.

write.block16

Esta función emplea a write.record16 para el envío de primitivas de salida de la monoputer a la nodriza. Esta función realiza la siguiente secuencia de llamado:

```
write.integer(WriteBlock16.Cmd)
write.integer(stream.id)
write.record16(longitud.registro, registro)
read.integer(resultado)
```

Initialize.graph

Esta funcion inicializa la nodriza en el modo gráfico de mayor resolución. Esta función realiza la siguiente secuencia de llamado:

```
write.integer(Initialize.Cmd)
read.integer(maxima.coord.x)
read.integer(drive.id)
read.integer(maxima.coord.y)
read.integer(resultado)
```

Los códigos de error son los siguientes:

0	No existe error.
-1	Modo gráfico no inicializado.
-2	No se cuenta con hardware para gráficas.
-3	No se tiene device driver.
-4	Device driver inválido.
-5	No hay suficiente memoria para el device driver.
-6	Modo gráfico inválido.
-7	otro.

Setgraphmode

Esta función cambia el modo de video que se tenga en la máquina nodriza. Esta realiza la siguiente secuencia de llamado:

```
write.integer(Setgraph.Cmd)
write.integer(graph.mode)
read.integer(maxima.coord.x)
read.integer(maxima.coord.y)
read.integer(resultado)
```

Los códigos de error corresponden a los descritos para Initialize.graph. La tabla 1 nos muestra los posibles modos para los diferentes identificadores de drive, los cuales son las que maneja el compilador de C de Borland.

graph.pixel

Esta función pone un pixel en la pantalla de la máquina nodriza, en las coordenadas dadas, y tiene la siguiente secuencia de llamado:

```
write.integer(Putpixel.Cmd)
write.integer(Coordenada.x)
write.integer(Coordenada.y)
write.integer(atributo)
read.integer(resultado)
```

Los códigos de error corresponden a los descritos para Initialize.graph.

Getpixel

Esta función lee el atributo del pixel, en una posición dada en la pantalla de la nodriza. Su secuencia de llamado es el siguiente:

```
write.integer(Getpixel.Cmd)
write.integer(Coordenada.x)
write.integer(Coordenada.y)
read.integer(atributo)
read.integer(resultado)
```

Los códigos de error corresponden a los descritos para Initialize.graph.

send.primitive

Esta función es empleada para el envío de la información acerca de las diferentes primitivas que se pueden generar. Esta función es muy importante por que permite mantener aislada la generación

de gráficas con el envio de estas. De esta manera, si el usuario desea graficar algunas otras primitivas de las que actualmente se tienen, o emplear algún otro algoritmo diferente a los que se usaron para generar las primitivas con que se cuenta, sólo tiene que hacer la rutina que la genera y emplear send.primitive para enviarla a la nodriza para que sea desplegada.

La secuencia de llamado para esta función es como sigue:

```
write.integer(Send.Prim.Cmd)
write.record16(xp)
write.record16(yp)
```

Donde yp y xp son dos bloques de enteros de 16 bits, con las coordenadas y y x de los puntos correspondientes a la primitiva a ser enviada.

En esta función no se regresa resultado, para dar una mayor velocidad cuando se va a desplegar alguna primitiva.

writescr

Esta función se encarga de enviar un bloque de bytes, el cual representa nuestra memoria de video del lado de la monoputer, para que sea desplegada en la nodriza. Su secuencia de llamado es el siguiente:

```
write.integer(WriteScreen.Cmd)
read.integer(maxima.coord.x)
read.integer(maxima.coord.y)
write.screen.rec(registro, maxima.coord.x, maxima.coord.y)
```

En esta función no se recibe mensaje de error, por que esto la haría muy lenta, pués se tendrían que hacer muchas revisiones.

de gráficas con el envio de estas. De esta manera, si el usuario desea graficar algunas otras primitivas de las que actualmente se tienen, o emplear algún otro algoritmo diferente a los que se usaron para generar las primitivas con que se cuenta, sólo tiene que hacer la rutina que la genera y emplear send.primitive para enviarla a la nodriza para que sea desplegada.

La secuencia de llamado para esta función es como sigue:

```
write.integer(Send.Prim.Cmd)
write.record16(xp)
write.record16(yp)
```

Donde yp y xp son dos bloques de enteros de 16 bits, con las coordenadas y y x de los puntos correspondientes a la primitiva a ser enviada.

En esta función no se regresa resultado, para dar una mayor velocidad cuando se va a desplegar alguna primitiva.

writescr

Esta función se encarga de enviar un bloque de bytes, el cual representa nuestra memoria de video del lado de la monoputer, para que sea desplegada en la nodriza. Su secuencia de llamado es el siguiente:

```
write.integer(WriteScreen.Cmd)
read.integer(maxima.coord.x)
read.integer(maxima.coord.y)
write.screen.rec(registro, maxima.coord.x, maxima.coord.y)
```

En esta función no se recibe mensaje de error, por que esto la haría muy lenta, pués se tendrían que hacer muchas revisiones.

readscr

Esta función se encarga de leer la memoria de video de la máquina nodriza y copiarla en un bloque de bytes del lado de la monoputer. Su secuencia de llamado es el siguiente:

```
write.integer(ReadScreen.Cmd)
read.integer(maxima.coord.x)
read.integer(maxima.coord.y)
read.screen.rec(registro, maxima.coord.x, maxima.coord.y)
```

En esta función no se recibe mensaje de error por que esto haría muy lenta dicha función pues se tendrian que hacer muchas revisiones.

clear

Esta función borra la pantalla, estando en alguno de los modos gráficos. Su secuencia es como sigue:

```
write.integer(Clear.Cmd)
read.integer(Result)
```

Los codigos de error corresponden a los descritos para Initialize.graph.

getmode

Esta función se emplea para obtener el valor que corresponde al modo gráfico actual. Su secuencia de llamado es el siguiente:

```
write.integer(GetMode.Cmd)
read.integer(Mode)
read.integer(Result)
```

Los códigos de error corresponden a los descritos para Initialize.graph.

getmaxx

Se emplea para obtener el valor de la máxima coordenada en el eje x, es decir, el número máximo de columnas que se puede manejar en el modo gráfico actual. Su secuencia de llamado es el siguiente:

```
write.integer(GetMaxx.Cmd)
read.integer(maxima.coord.x)
read.integer(Result)
```

Los códigos de error corresponden a los descritos para Initialize.graph.

getmaxy

Se emplea para obtener el valor de la máxima coordenada en el eje y, es decir, el número máximo de renglones que se puede manejar en el modo gráfico actual. Su secuencia de llamado es el siguiente:

```
write.integer(GetMaxx.Cmd)
read.integer(maxima.coord.y)
read.integer(Result)
```

Los códigos de error corresponden a los descritos para Initialize.graph.

graphterminate

Esta función termina con el modo gráfico y nos regresa al modo de texto. Esta puede emplearse cuando se ha terminado de realizar gráficas. Su secuencia de llamado es la siguiente:

```
write.integer(Termgraph.Cmd)
read.integer(Result)
```

get.col.prim

Esta función es empleada para conocer el color con que se van a graficar las primitivas de salida, cuando éstas son enviadas directamente, para ser desplegadas. Su secuencia es como sigue:

```
write.integer(GetColprim.Cmd)
read.integer(color)
read.integer(result)
```

Cada color tiene un número que lo identifica, entre los posibles colores que se tienen para una paleta dada de colores [13].

set.col.prim

Esta función sirve para elegir alguno de los posibles colores que se tienen disponibles, en el modo gráfico en el que se esté en el momento en que se ejecuta, para graficar las primitivas. Su secuencia es la siguiente:

```
write.integer(SetColPrim.Cmd)
write.integer(color)
read.integer(result)
```

Los códigos para las funciones que accesan a MyAFserver, para gráficar, son los siguientes:

WriteBlock16.Cmd	25
Initialize.Cmd	26
Setgraph.Cmd	27
Putpixel.Cmd	30
Send.Prim.Cmd	31
ReadScreen.Cmd	640
WriteScreen.Cmd	642
Clear.Cmd	35
GetMode.Cmd	36
GetMaxx.Cmd	37
GetMaxy.Cmd	38
Termgraph.Cmd	41
GetColPrim.Cmd	42
SetColPrim.Cmd	43

2.3- EL PROGRAMA INTERFAZ DEL LADO DE LA MAQUINA NODRIZA.

Ahora describiremos el programa MYAFServer, el cual está escrito en lenguaje C y es el que se encarga de atender las peticiones provenientes de la monoputer, cuando esta desea hacer uso de los recursos de la nodriza.

Este programa está formado básicamente de un conjunto de rutinas encargadas de recibir y enviar datos y bloques de datos desde y hacia la monoputer. De esta manera es posible mantener la comunicación entre las dos máquinas, y además realizar las operaciones necesarias solicitadas por la monoputer, para accesar los recursos de la nodriza.

La comunicación es sincrona y funciona de la siguiente manera:

El programa MYAFServer constantemente está revisando si la transputer tiene listo algún dato, lo que significaría que ésta desea enviarle el código de alguna función para su ejecucion del lado de la nodriza (puerto el puerto LINK.IN.STATUS). Entonces MYAFserver sabe que enseguida la monoputer enviará el código de la función que se desea accesar, a través del puerto LINK.READ, de acuerdo al protocolo básico y de acuerdo al protocolo de secuencia de llamado para las diferentes funciones. Es decir, que espera que la monoputer envie un entero, el cual es el código de la función que la monoputer desea accesar. Después, según sea el código del comando recibido, la monoputer indica por medio de LINK.IN.STATUS, que tiene lista para enviar una serie de datos y bloques de datos como parámetros a través del puerto LINK.READ; posteriormente indica, por medio de LINK.OUT.STATUS, que espera le sean devueltos otra serie de parámetros a través de LINK.WRITE.

Para finalmente indicar que espera le sea enviado un entero con el código de error, enviándole el resultado de la operación.

Además de realizar estas operaciones, MyAFserver tiene la capacidad de cargar un programa ejecutable, desde la máquina nodriza a la memoria de la monoputer, y de inicializar ésta última.

2.4.- LOS ENCABEZADOS DE LAS FUNCIONES DEL TERCER NIVEL

Enseguida describimos la forma de emplear las funciones del tercer nivel, explicando qué es cada párametro y su tipo mediante sus encabezados.

```
PROC term.graph (CHAN OF ANY from.filer, to.filer, INT result)
```

from.filer y to.filer son los canales mediante los cuales el proceso que emplea esta función se comunica con Myfilt. Esto se aplica al resto de las funciones que tienen a estos canales como parámetros. result es una variable en la cual se devuelve el valor del código de error que indica el resultado de la operación.

```
PROC write.record16 (CHAN OF ANY to.filer,INT len,VAL []INT16 r)
```

len es una variable en la que se obtiene la longitud del registro de datos que se envia a la nodriza. r es el arreglo de enteros que se desea enviar.

```
PROC write.block16 (CHAN OF ANY from.filer, to.filer,
                    VAL INT stream.id, VAL []INT16 record,
                    INT len, result)
```

stream.id debe tener el identificador del stream a emplear. record es el bloque de datos que se desea enviar. len tiene la longitud del bloque que se envia y en result obtenemos el resultado de la operación.

```
PROC Initialize.graph(CHAN OF ANY from.filer, to.filer,
                      INT driver.id, maxx, maxy, result)
```

driver.id es el código del driver empleado por la nodriza en el modo gráfico en que se inicializa [13]; maxx y maxy son variables en las que se obtienen las máximas coordenadas posibles, en el modo gráfico, en el cual se inicializa el sistema, el cual corresponde a la máxima resolución que permite la máquina nodriza. result es el parámetro del cual se obtiene el código de error que corresponde al resultado de la operación.

```
PROC Setgraph.mode(CHAN OF ANY from.filer, to.filer,
                   INT graphmode, maxx, maxy, result)
```

graphmode es el parámetro mediante el cual indicamos el nuevo modo gráfico, al cual queremos cambiar y el cual depende del driver para gráficas que se tenga; teniendo valores de cero en adelante para cada una de ellos, siendo cero el de la resolución más baja para cada caso. maxx y maxy son los valores máximos de las coordenadas x e y respectivamente. En result obtenemos información sobre lo sucedido con esta función, y sus codigos son los mismos que para Initialize.graph.

```
PROC graph.pixel(CHAN OF ANY from.filer,to.filer,
                 VAL INT x,y,value,INT result)
```

x & y son los parámetros mediante los cuales indicamos las coordenadas x & y donde se graficará el punto. value se emplea para indicar el atributo del pixel a gráficar y result nos devuelve el resultado de la operación.

```
PROC get.pixel(CHAN OF ANY from.filer, to.filer,
               INT x, y, value, INT result)
```

x & y son las coordenadas del pixel del cual queremos obtener el valor de sus atributos, el cual obtenemos en value. result es igual que en las anteriores funciones.

```
PROC send.primitive(CHAN OF ANY from.filer, to.filer,
                    []INT16 xp,[]INT16 yp, INT i)
```

xp y yp son los arreglos de enteros de 16 bits conteniendo las corrdenadas x & y respectivamente, de la primitiva que se desea gráficar. i debe tener el valor del número de puntos que tiene dicha primitiva.

```
PROC read.screen.rec(CHAN OF ANY from.filer,
                     [] [] BYTE record, INT lenx, leny)
```

record contiene uno de los bloques que representan a la pantalla. lenx y leny deben tener los valores de las máximas coordenadas de pantalla en el modo gráfico actual.

```
PROC readscr (CHAN OF ANY  from.filer,to.filer,
              [][] BYTE record, INT lenx, leny)
```

los parámetros de esta función son los mismos que para read.screen.rec.

```
PROC write.screen.rec(CHAN OF ANY to.filer,
                      [] [] BYTE record, VAL INT lenx, leny)
```

record contiene uno de los bloques que representan a la pantalla. lenx y leny deben tener los valores de las máximas coordenadas de pantalla en el modo gráfico actual.

```
PROC writescr (CHAN OF ANY  to.filer,
               [][] BYTE record, VAL INT lenx, leny)
```

los parámetros de esta función son los mismos que para read.screen.rec.

```
PROC clear(CHAN OF ANY from.filer, to.filer, INT result)
```

result es igual que en Initialize.graph.

```
PROC getmode(CHAN OF ANY from.filer, to.filer, INT mode, result)
```

mode es el parámetro en el que obtenemos el modo gráfico en que se encuentra el sistema en el momento de ejecutar este procedimiento, sus valores corresponden a los descritos para setmode. result es igual que para Initialize.graph.

```
PROC getmaxx(CHAN OF ANY from.filer, to.filer, INT maxx, result)
```

maxx es el parámetro en que obtenemos el valor correspondiente a la máxima coordenada en el eje x, en el modo gráfico actual. result es igual que para Initialize.graph.

```
PROC getmaxy(CHAN OF ANY from.filer, to.filer, INT maxy, result)
```

maxy es el parámetro en que obtenemos el valor correspondiente a la máxima coordenada en el eje y, en el modo gráfico actual. result es igual que para Initialize.graph.

```
PROC get.col.prim(CHAN OF ANY from.filer, to.filer,
                  INT colorp, result)
```

colorp es el color con que se grafican las primitivas de salida, mientras no se cambie este valor mediante set.col.prim.

```
PROC set.col.prim(CHAN OF ANY from.filer, to.filer,
                  VAL INT colorp, result)
```

colorp es el color con que se graficaran las primitivas de salida desde el momento de ejecutar esta función y hasta que no se cambie este valor mediante esta misma función.

2.5.- LA LIBRERIA DE FUNCIONES DE GENERACION DE GRAFICAS

Además de lo anterior, se ha creado una pequeña libreria que sirve para la generación de gráficas, la cual puede ser fácilmente modificada para las necesidades especificas del usuario ya que los cambios en esta libreria no afectan al resto de la interfaz. Enseguida describimos las rutinas con que se cuenta en esta libreria.

```
PROC evaluate.line(VAL INT16 x1, y1, x2, y2,
                   []INT16 xp, []INT16 yp,INT i)
```

Esta función se emplea para generar las coordenadas de una linea, y se basa en el método de Bresenham [11].Sus parámetros son como sigue: x1, y1, x2, y2, son las coordenadas de los puntos extremos de la recta a calcular. Las coordenadas de los puntos de la recta son devueltas en los arreglos xp y yp, cuyo número se obtiene en el parametro i.

```
PROC evaluate.circle(VAL INT16 xc, yc, r,
                     []INT16 xp, []INT16 yp, INT i)
```

Esta función se emplea para generar las coordenadas de una circunferencia, y se basa en el método de Bresenham [11]. x & y son las coordenadas del centro y r el radio de la circunferencia. Las coordenadas de los puntos de la circunferencia son devueltas en los arreglos xp y yp, cuyo número es obtenido en el parametro i.

```
PROC writefig.screen(INT i,[]INT16 xp,[]INT16 yp ,
                     [][]BYTE screen)
```

Esta función es empleada para escribir alguna primitiva evaluada previamente, en el area de memoria de la monoputer en donde mantenemos la información que representa una pantalla de video. i es el número de punto que representan a la primitiva. xp y yp son los arreglos con los puntos de la primitiva; y screen es la matriz que representa nuestra pantalla de video.

```
PROC putpixloc.screen (VAL INT16 x, y, [][] BYTE screen)
```

putpixel.screen se empleada para escribir un punto en el area de memoria de la monoputer donde mantenemos la información que representa una pantalla de video. x & y son las coordenadas del punto a escribir. screen es igual que para writefig.screen.

```
PROC transforma(VAL REAL32 ptox,ptoy,ptoz, VAL []REAL32 e.norm,
                VAL []REAL32 ip,  VAL []REAL32 jp, []REAL32 ptof,
                INT16 rx, ry)
```

transforma es empleada para obtener la proyección de un punto en tres dimensiones.
ptox, ptoy y ptoz, son las coordenadas x, y, z respectivamente del punto a graficar. e.norm es el vector perpendicular al plano de proyección; ip es el vector unitario paralelo al eje x de nuestro plano de proyección; jp es el vector unitario paralelo al eje y de nuestro plano de proyección; ptof es vector de posición de nuestro punto de fuga; rx y ry son las coordenadas x & y, transformadas de nuestro punto sobre el plano de proyección.

El plano de proyección representa para nosotros la pantalla de la nodriza.

```
PROC Desp.Net(CHAN OF ANY from.filer, to.filer, [][][]REAL32 s,
              INT m,n,[]REAL32 enormal, []REAL32 pfuga, []REAL32 ip,
              []REAL32 jp, []INT16 xp, []INT16 yp)
```

Desp.Net es empleada para desplegar una malla de puntos en tres dimensiones sobre la pantalla. Sus párametros son como sigue:

s es la matriz con las coordenadas de nuestra malla; m y n son el número de punto por lado de nuestra malla, de tal manera que nuestra malla tiene nxm puntos. Al desplegarse la malla, sus puntos son unidos por lineas para darle un verdadero aspecto de malla de alambre. enormal, pfuga, ip y jp son igual que para transforma. xp y yp son un par de arreglos empleados en forma interna por Desp.Net.

```
PROC fast.write..screen(CHAN OF ANY from.filer, to.filer,
                        []BYTE Scree  INT result)
```

Esta función realiza lo mismo que writescr, descrita anteriormenete. La diferencia con ésta es que fast.write.screen es más rápida. Sin embargo para lograr esta mayor rapidez, sacrifica la portabilidad, ya que esta hecha especialmente para la tarjeta (CGA) con que cuenta la máquina nodriza en que se encuantra. Para realizar esta tarea con otras tarjetas es conveniente usar writescr.

Screen es el arreglo de bytes que nos sirve para almacenar la información sobre la pantalla.

```
PROC fast.read.screen(CHAN OF ANY from.filer, to.filer, []BYTE Screen,
                      INT result)
```

Esta función realiza la misma tare aque readscr, descrita

anteriormenete. La diferencia con ésta es que fast.read.screen es más rapida. Sin embargo al igual que fast.write.screen, sacrifica la portabilidad, por la misma razón. Para realizar esta tarea con otras tarjetas es conveniente usar readscr.

Screen es el arreglo de bytes que nos sirve para almacenar la pantalla.

Las rutinas de esta libreria emplean ciertos algoritmos. Sin embargo, si el usuario no desea estos algoritmos, puede hacer los suyos y sustituir los originales. Esto es posible gracias a que la interfaz esta hecha de tal manera que esta libreria no afecte al resto.

```
PROC fastp(VAL INT columna, renglon, []BYTE Screen,
           VAL []INT Ys)
```

Esta función se emplea junto con fast.write.screen y fast.read.screen, y se emplea para poner un pixel en el area de memoria en que se almacena la pantalla.

columna es la coordenada x y renglon la coordenada y, en donde se pondrá el pixel; Ys es un arreglo auxiliar de enteros el cual debe ser evaluado antes de emplear fastp, mediante evalys.

```
PROC write.fast.fig(INT i,[]INT16 xp, []INT16 yp,
                    []BYTE screen, []INT Ys)
```

Esta función se emplea con fast.write.screen y fast.read.screen, y sirve para almacenar una primitiva completa en el area de memoria que nos representa la pantalla; xp & yp son los arreglos en los que se tienen las coordenadas x e y de la primitiva, i es el número de puntos que tenemos de esa primitiva; Ys es como en fastp.

PROC evalys([]INT Ys)

Esta función debe ser ejecutada antes de emplear fastp o write.fast.fig, ya que evalua algunos valores que éstas necesitan.
Ys es el arreglo que se evaluará.

CAPITULO 3
LA INTERFAZ COMO RECURSO DE LA NODRIZA

En muchas ocasiones el usuario tiene que hacer gráficas en alguna parte de su programa, sin embargo, si desea emplear a la monoputer para esto, tendría que escribir su programa completamente para que corriera en la monoputer, es decir, en OCCAM [1],[2]; o adquirir el compilador que desee emplear para esta máquina; esto posiblemente no es muy ventajoso para el usuario y su aplicación; esta parte de la interfaz ofrece una solución a este problema.

Fundamentalmente esta parte del trabajo consiste de un conjunto de programas encargados de permitir el acceso a la monoputer por cualquier programa de usuario que corra del lado de la nodriza, de tal manera que este vea a la monoputer como un recurso más de la nodriza.

3.1.-COMO ACCESAR A LA MONOPUTER DESDE UN PROGRAMA DE USUARIO EN LA NODRIZA.

El acceso a la monoputer se logra mediante la generación por software de la interrupción 65h, desde el programa que la quiera accesar. Los párametros que se deseen pasar a la monoputer deben ser cargados en un arreglo de enteros de la siguiente manera:

El primer elemento del arreglo siempre se carga con el número de la función que se desea ejecutar. Los siguientes elementos del arreglo contendrán los párametros que necesita la función que se desea ejecutar.

Antes de generar la interrupción es necesario cargar la dirección inicial del arreglo con los párametros en los registros AX y BX; AX con el segmento y BX con el offset; también se debe cargar el número de párametros (sin contar el número de función) en el registro CX.

Para que algún programa de usuario accese a la monoputer, se debe realizar la siguiente secuencia:

- Debe hacer una solicitud a la monoputer enviándole el código de función que desea que realice, así como sus párametros, en la forma que ya se explicó, y a continuación generar la interrupción 65h.

- Si la monoputer tiene que accesar a la nodriza para ejecutar la función que le fue solicitada, el programa del usuario tiene la capacidad de hacer que la monoputer espere hasta que éste le de permiso. Para hacer esto, el programa de usuario debe simplemente cargar al registro CX, con un cero, y generar la interrupcion 65h.

Actualmente la mayoría de los compiladores comerciales para trabajar en máquinas del tipo de la nodriza, permiten generar interrupciones, lo que da oportunidad al usuario de emplear a la monoputer através de esta parte de la interfaz sin tener que programar en OCCAM; salvo en la ocación en que tenga que realizar tareas específicas no contempladas en esta parte del trabajo, en cuyo caso solamente hay que crear la rutina que realice el trabajo especifico, lo que no ofrece dificultad especial. Y de ahí en adelante vería a la monoputer como una caja negra.

3.2.- DESCRIPCION DE LA ESTRATEGIA SEGUIDA

La comunicación entre la monoputer y la nodriza, en esta parte, se ilustra en las figuras 7 y 8, y se describe enseguida:

Del lado de la monoputer existe un proceso llamado TServer,el cual revisa constantemente si existe una solicitud por parte de la nodriza para accesar a la monoputer. Este proceso espera hasta que haya una solicitud, entonces recibe el número de la función que se solicita y sus correspondientes párametros. Enseguida ejecuta la operación solicitada , pidiendo a la nodriza le de permiso para accesar los recursos de esta máquina, si es que es necesario; espera hasta que se le de permiso para hacerlo, después de lo cual su trabajo ha terminado por el momento y vuelve a revisar si existe una nueva solicitud por parte de la nodriza para accesar a la monoputer.

Ahora bien, del lado de la nodriza las cosas suceden como sigue: cuando un programa de usuario solicita que la monoputer realice una operación, transfiere el control a un programa residente llamado Server2, para que sea éste el ancargado de entablar la comunicación con la monoputer, de acuerdo con los protocolos ya descritos anteriormente.

Este Server2 es el que se encarga de enviar la solicitud, el código de función y sus párametros a la monoputer. Una vez hecho esto, se regresa el control al programa del usuario. Si la función que fue solicitada necesita accesar los recursos de la nodriza, el programa de usuario podrá en cualquier momento dar permiso para que se haga el acceso. Para hacer esto, el usuario transfiere el control nuevamente al programa Server2, para que este sea quien dé el permiso a la monoputer, y para que realice las operaciones correspondientes solicitadas para accesar los diferentes recursos de la nodriza.

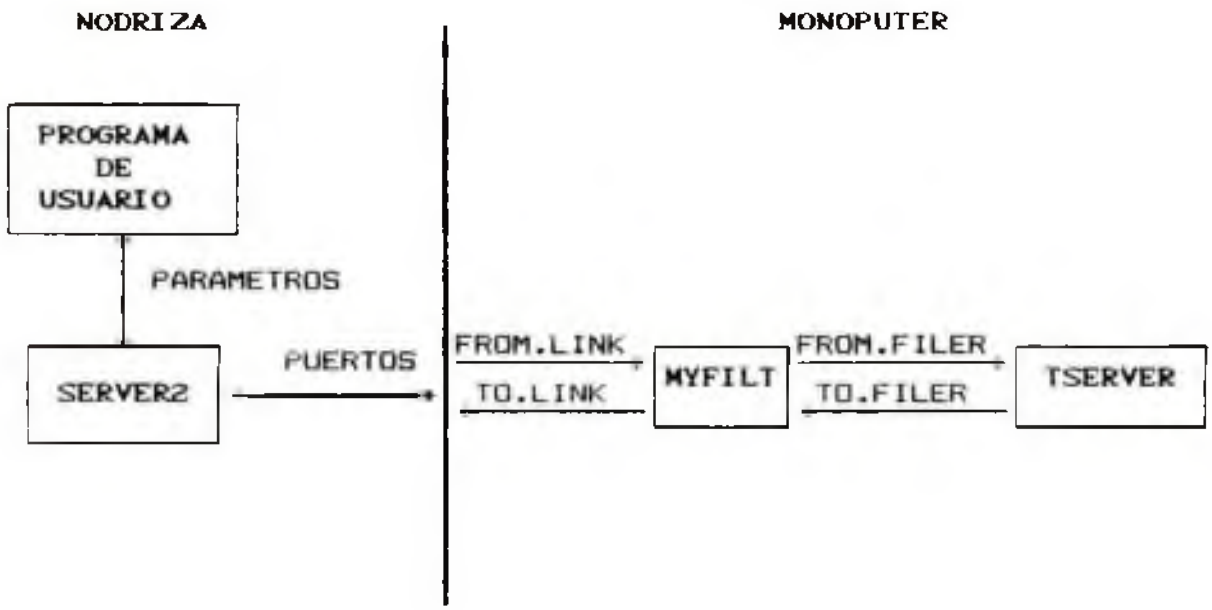


Fig. 7. Estructura de la interfaz como recurso de la nodriza.

Tanto el proceso MyFilt como el TServer, corren en forma paralela y son creados por el proceso Myharn, el cual termina una vez que ha finalizado con esta tarea. El proceso MyFilt es el mismo que el que se usó para la primera parte de la interfaz. Tserver tiene un ciclo infinito para mantener la información generada en diferentes accesos consecutivos a la monoputer. Esta

comunicación entre las dos máquinas, a través de los programas que corren en ambas máquinas, puede verse en la figura 8.

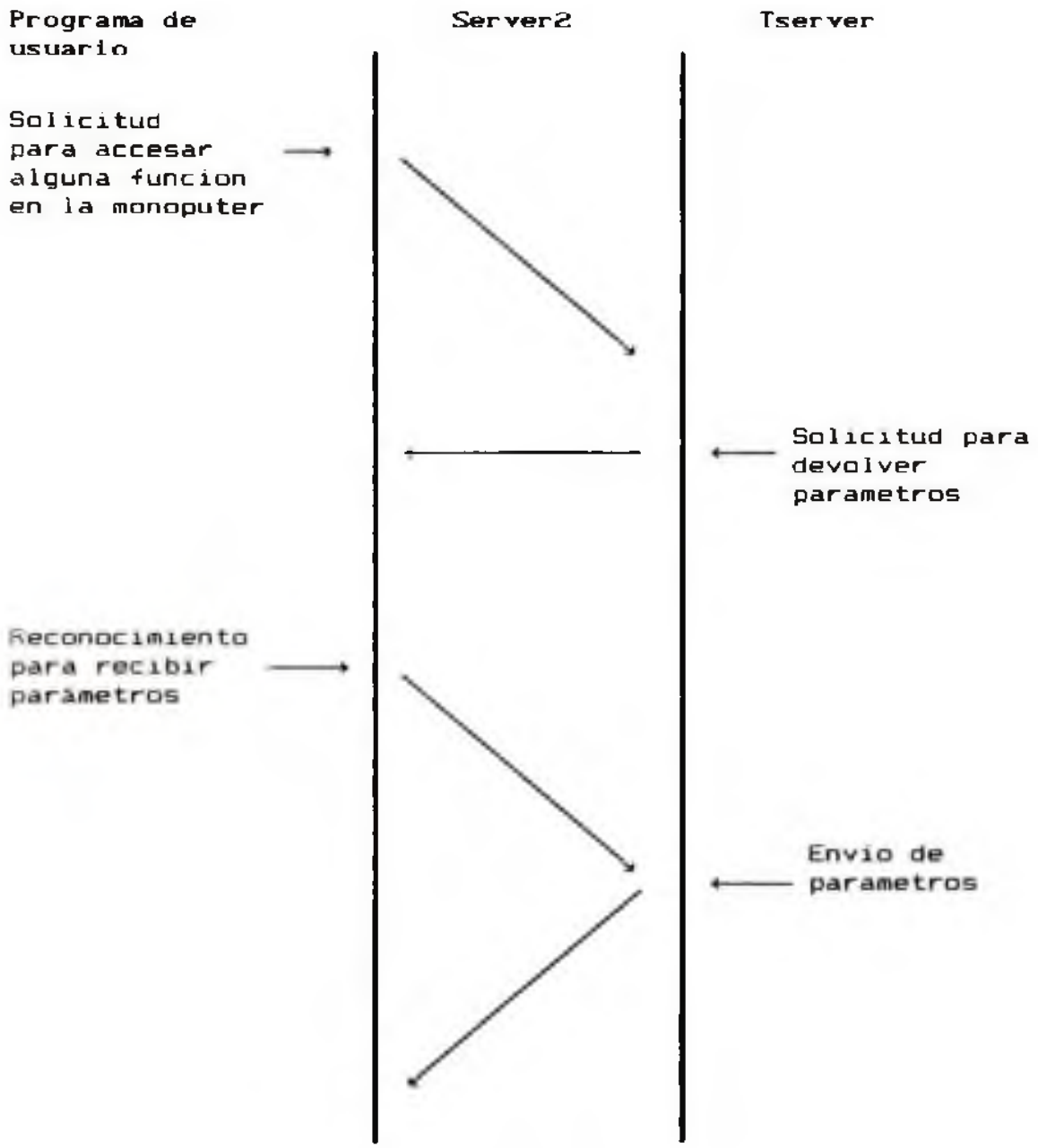


Fig. 8. Tranferencia de datos, para accesar a la monoputer.

Para reinicializar a Tserver, es necesario cargar nuevamente a la monoputer después de rinicializarla, todo esto empleando el programa Load, del lado de la nodriza.

Hay que hacer notar que las funciones que puede solicitar el programa del usuraio a la monoputer, son sólo aquellas en las que resulta más ventajoso que intervenga esta máquina, ya que ciertas funciones resultan más eficientes si las realiza directemente el programa del usuario.

CAPITULO 4
LA INTERFAZ COMO UN SERVIDOR RESIDENTE

La mayoría de las aplicaciones en las que se empleará la monoputer son las que necesitan una gran cantidad de cálculos, lo que la mantendrá ocupada por un buen tiempo. Sin embargo, en todo este tiempo la nodriza estará ociosa. Para solucionar esto se ha hecho el trabajo necesario para que la interfaz desocupe a la nodriza mientras la monoputer no necesite sus recursos. En esta parte describimos cómo se logró esto.

Esta parte es en sí un servidor residente para la máquina monoputer. Este servidor se encarga de atender a la monoputer cuando esta desee accesar los recursos de la nodriza, suspendiendo en forma temporal el proceso que esté ejecutandose en el momento en el que la monoputer indique al servidor que desea accesar a la nodriza. Enseguida describimos este servidor y su funcionamiento.

4.1-DESCRIPCION DE LA ESTRATEGIA SEGUIDA

Cuando este servidor es ejecutado, el mismo se instala como un programa residente en memoria, momento a partir del cual comienza su trabajo, no permaneciendo en forma pasiva sino trabajando en forma pseudo paralela con el proceso que se encuentre en ejecución en la máquina nodriza.

En esta parte la comunicación entre la monoputer y la nodriza se realiza como se describe enseguida.

Del lado de la nodriza se encuentra el servidor residente, el cual se divide en dos partes:

La primera parte la forman básicamente tres rutinas, dos de las cuales constantemente revisan si la monoputer desea accesar a la nodriza; una de ellas es la encargada de atender a la monoputer mientras se esté ejecutando algún otro proceso; la otra se encarga de lo mismo, pero mientras el sistema operativo tiene el control y está esperando que se le dé algún comando. La otra rutina es la encargada de realizar el cambio de contexto,con objeto de que entre en acción el servidor gráfico y atienda a la monoputer. Estas rutinas se encargan de mantener el control y evitar que se intente accesar recursos cuando pueda ser peligroso. Esto es importante por que cuando la monoputer desea accesar a la nodriza ésta debe tener la libertad de accesar los recursos de ésta, por lo que sería peligroso permitirle hacer esto, si es que algún otro proceso estuviera manejando dichos recursos en el momento en el que la monoputer los solicitara . De esta manera, estas dos rutinas se encargan también de la seguridad.

Una vez que la monoputer indica que desea accesar a la nodriza, que una de la dos rutinas mencionadas se ha dado cuenta de esto y además de que no existe ningún peligro, estas rutinas se encargan de hacer un cambio de contexto, suspendiendo el proceso que en ese momento esté corriendo y despiertan al servidor gráfico. En este cambio de contexto se guarda la información más importante acerca del proceso que se suspende. De esta manera, al regresar el control al programa suspendido, éste seguirá trabajando como si no hubiera existido el cambio.

La tercera parte es en sí el servidor, el cual se encargará de atender a la monoputer de una forma parecida a como lo viene realizando el servidor descrito en la primera parte, el cual será en sí quien se encargará del resto del trabajo.

El dibujo de la firura 9 nos ayudará a comprender mejor todo esto.

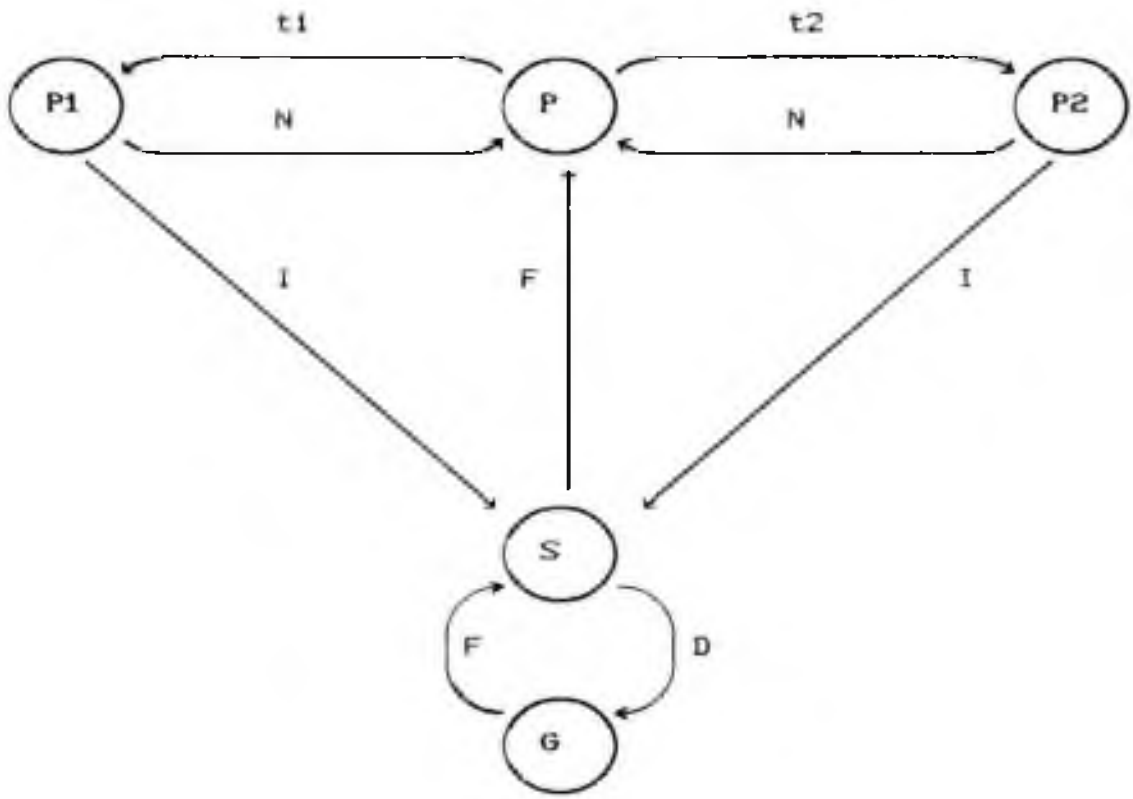


Fig. 9. Flujo de control entre los procesos de la interfaz residente.

En donde tenemos que:

Los círculos representan rutinas o procesos que corren en la nodriza, y las lineas, condiciones que se dan para que el control pase de una rutina o proceso a otro. Estos son como se indica enseguida.

P1.- Es uno de los procesos que se encargan de revisar constantemente si la monoputer indica que necesita accesar a la nodriza; esta verificación la lleva a cabo cada tiempo t1.

P2.- Es el otro proceso encargado de revisar constantemente si la monoputer indica que necesita accesar a la nodriza cada tiempo t2.

P.- Es el proceso que está trabajando actualmente en la nodriza.

S.- Es la rutina encargada de hacer el cambio de contexto para despertar al servidor.

G.- Representa al servidor.

t1.- Es la condición, que se cumple cuando ha transcurrido el tiempo determinado para que se ejecute la primera rutina que revisa, si la monoputer está indicando que necesita accesar a la nodriza.

t2.- Es la condición, que se cumple cuando ha transcurrido el tiempo determinado para que se ejecute la segunda rutina que revisa, si la monoputer está indicando que necesita accesar a la nodriza.

N.- Es la condición que se cumple cuando se ha deshabilitado al servidor; esto sucede cuando es peligroso despertarlo o cuando la monoputer no necesita accesar a la nodriza y el servidor no debe estar despierto en ese momento.

I.- Es la condición que se cumple cuando la monoputer desea accesar a la nodriza.

D.- Esta condición siempre se cumple y es cuando se realiza el cambio de contexto para despertar al servidor.

F.- Esta condición se cumple una vez que la monoputer ha terminado con la función que accesó del servidor.

En esta parte el servidor no se encarga de cargar los programas en la monoputer, sino sólo de atender dicha máquina. Por lo tanto, se hace necesario un programa especial, para realizar esta operación y la de resetear a la monoputer, por lo que se creó el programa Load, que se encarga de estas tareas, además de evitar que se despierte al servidor mientras no se haya cargado algún programa a la monoputer, evitando así que éste sea despertado cuando no debe. Este programa se protege a si mismo de ser interrumpido por el servidor gráfico, ya que resultaría muy trágico que esto sucediera mientras se está cargando un programa a la monoputer. Esto es debido a que el servidor gráfico utiliza los mismos puertos para comunicarse con la monoputer que el programa Load.

Al igual que Load, cualquier programa que esté corriendo en la máquina nodriza, y que tenga la necesidad de no ser interrumpido por la monoputer, cuando ésta desea accesar a la nodriza, tiene la capacidad de deshabilitar al servidor residente. Esto lo debe hacer de la siguiente manera: primero

debe cargar al registro AX con un 1, y enseguida generar la interrupción 64h. Si desea habilitarlo nuevamente, debe también generar la interrupción 64h, pero en este caso debe previamente cargar AX con un cero.

Del lado de la monoputer todo es como se manejó en la primera parte de la interfaz.

En la firura 10 se muestra la forma en que se comunican ambas máquinas en esta parte de la interfaz.

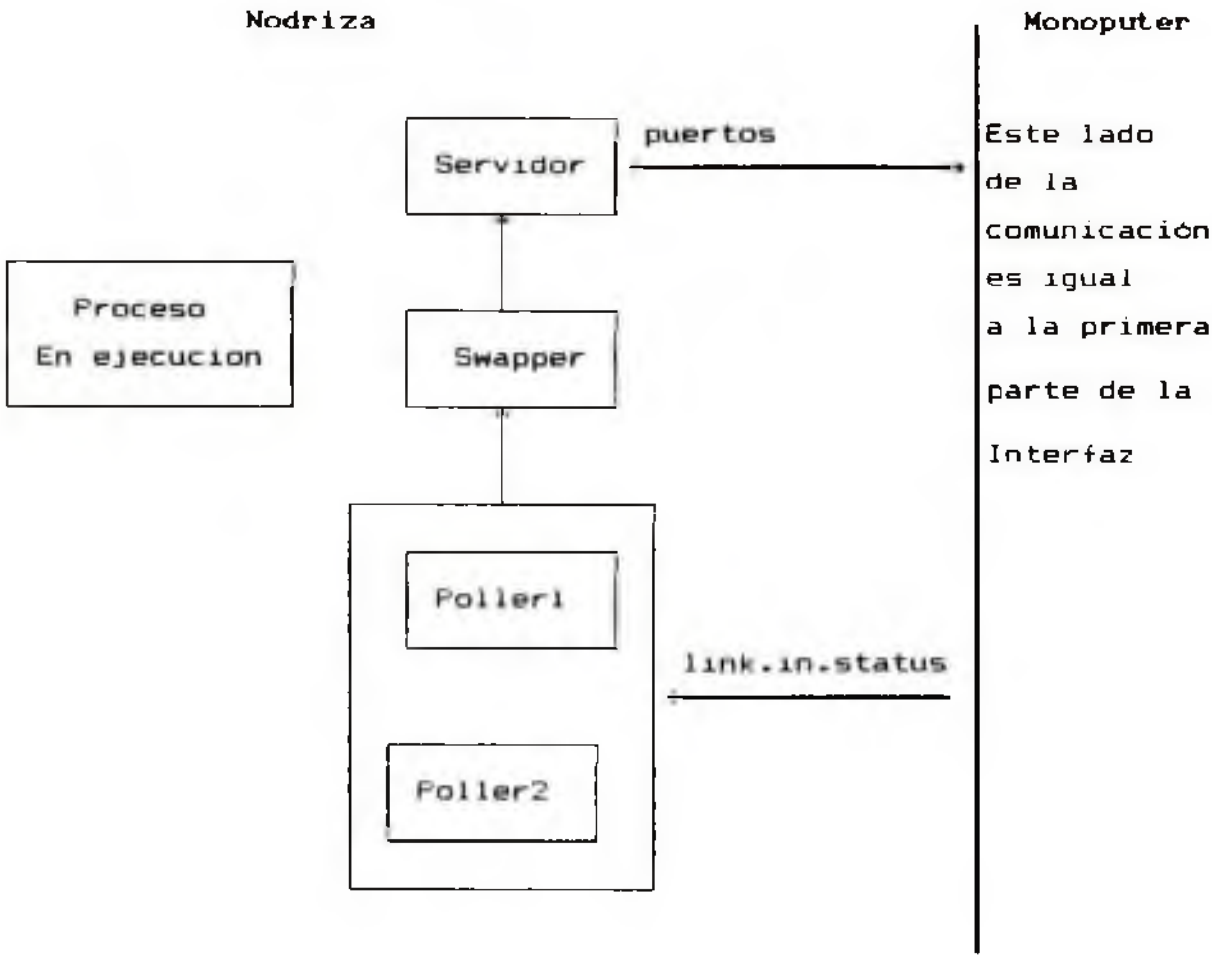


Fig. 10. *La extructura de la interfaz residente.*

Una vez que se ha instalado el servidor, y después de que se haya cargado algún programa a la monoputer empleando Load, no es conveniente ni compilar programas en Occam, ni tampoco emplear el servidor anterior, ya que para estas operaciones se emplean los

mismos puertos y canales de comunicación. por lo que seguramente habría problemas. Para solucionar este problema. se sugiere no emplear al mismo tiempo estos servidores. Sin embargo. en caso de que se desee usar el servidor normal puede llegar a realizarse teniendo cierto cuidado, y deshabilitando de manera temporal al servidor residente, de la misma manera en que el programa load lo hace.

También hay que tener en cuenta que el usuario no podrá modificar los vectores de interrupción, del 64h al 66h, ya que éstas son ocupadas por el servidor.

CAPITULO 5
EJEMPLOS DE APLICACION

Se ha realizado un ejemplo empleando la interfaz para apreciar como trabaja ésta, y de esta manera tener alguna forma de hacer una comparación con la creación de gráficas, empleando la monoputer y otra máquina, que en este caso fué la AT, que funciona como nodriza.

El ejemplo, trata de unas superficies de Bezier [12], [10]; el cual fué elegido, pues para ello se consume una gran cantidad de tiempo en cálculos matemáticos. Para realizar la comparación se graficó una superficie de 24x24 puntos, los cuales se unieron mediante rectas para dar un mejor aspecto, a partir de un conjunto de 4x4 puntos. Esto se hizo empleando el mismo algoritmo, tanto en la AT como en la monoputer. Para esta última se escribió el programa. El programa, para correr en la AT, fué escrito en lenguaje C, y el programa, para correr en la transputer, fué escrito en OCCAM. Prácticamente todos los cálculos para generar la grafica completa son ejecutados en la máquina en que se prueba cada uno de los dos programas, es decir, tanto para generar los puntos de la superficie como los puntos de las rectas que los unen.

La gráfica es la misma para ambas máquinas, y es la que se muestra en la fig.9 de la siguiente página, la cual tiene forma de silla.

Más adelante se muestran algunos otros dibujos. Se trata de un trocito de membrana cuadradá vibrando, graficado en diferentes momentos (fig. 11 y 12); también se muestra la silla pero ahora calculando 48x48 puntos (fig. 13) y dos curvas fractales [11],(fig. 14 y 15).

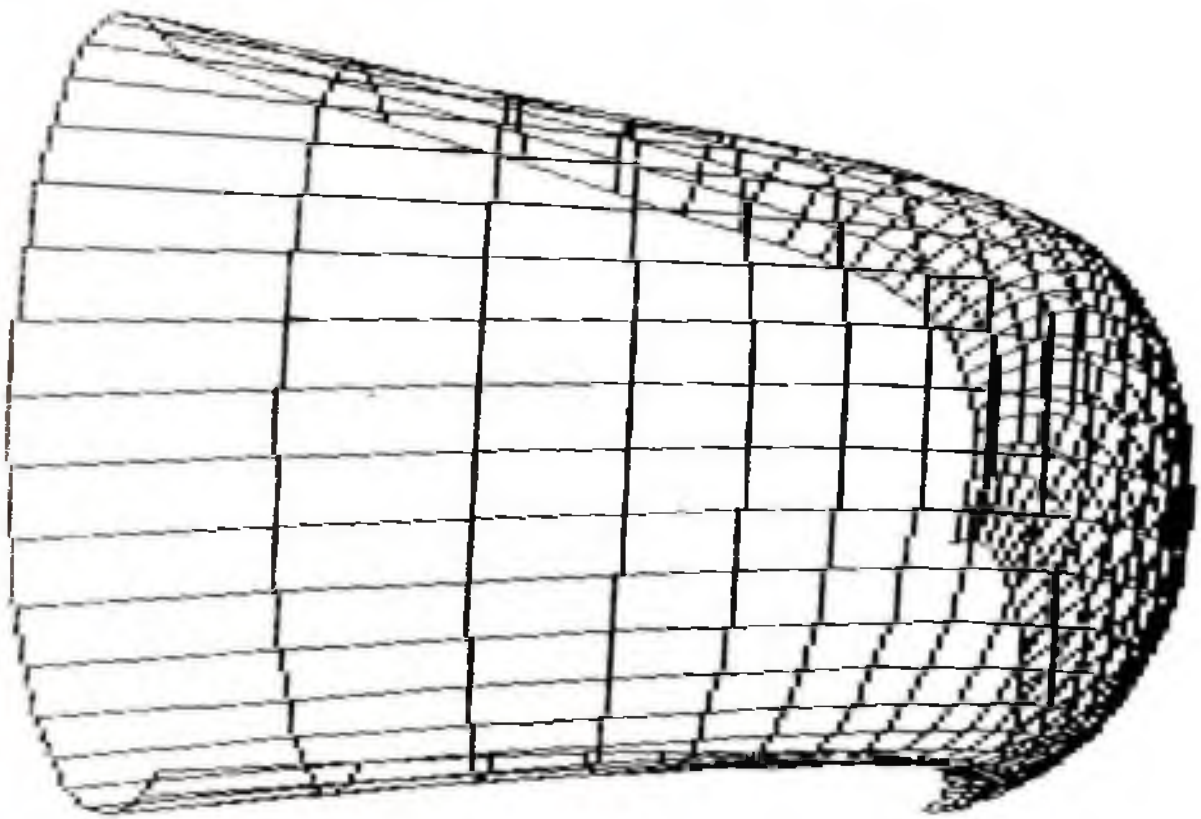


Fig. 11.

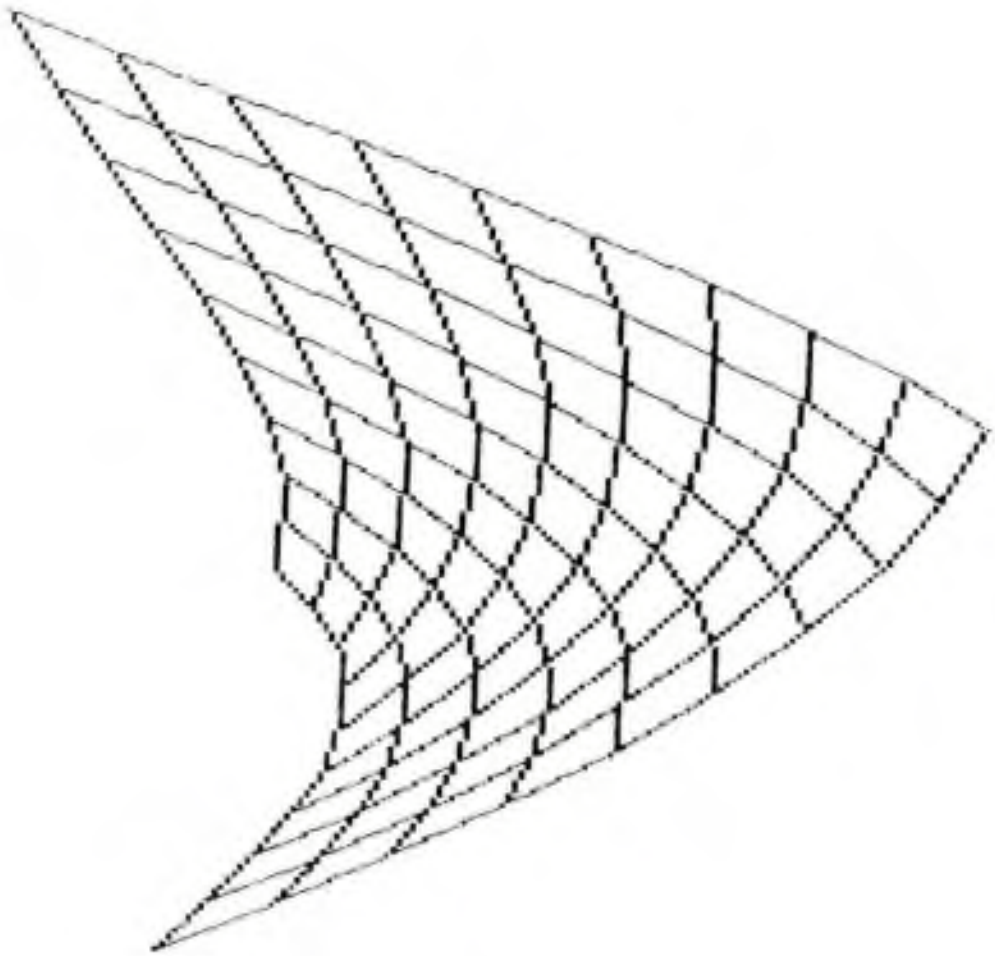


Fig. 12.

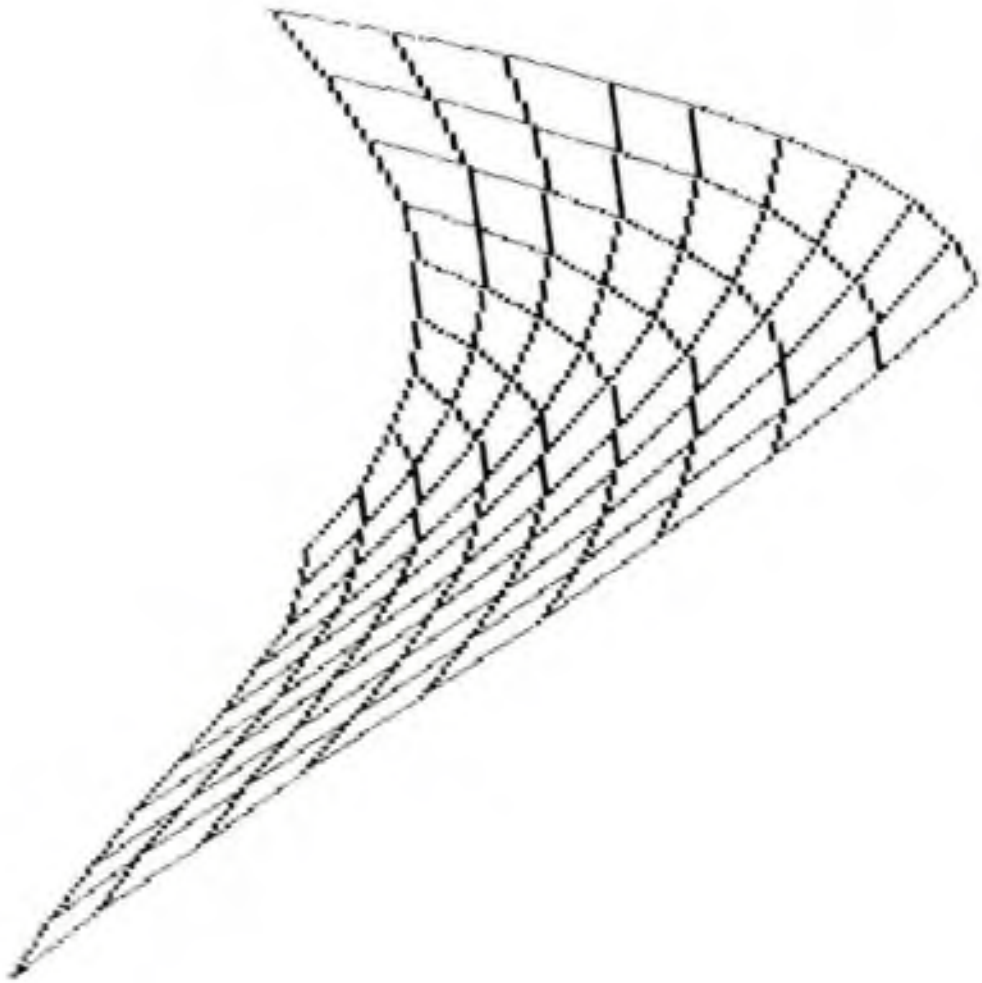


FIG. 13.

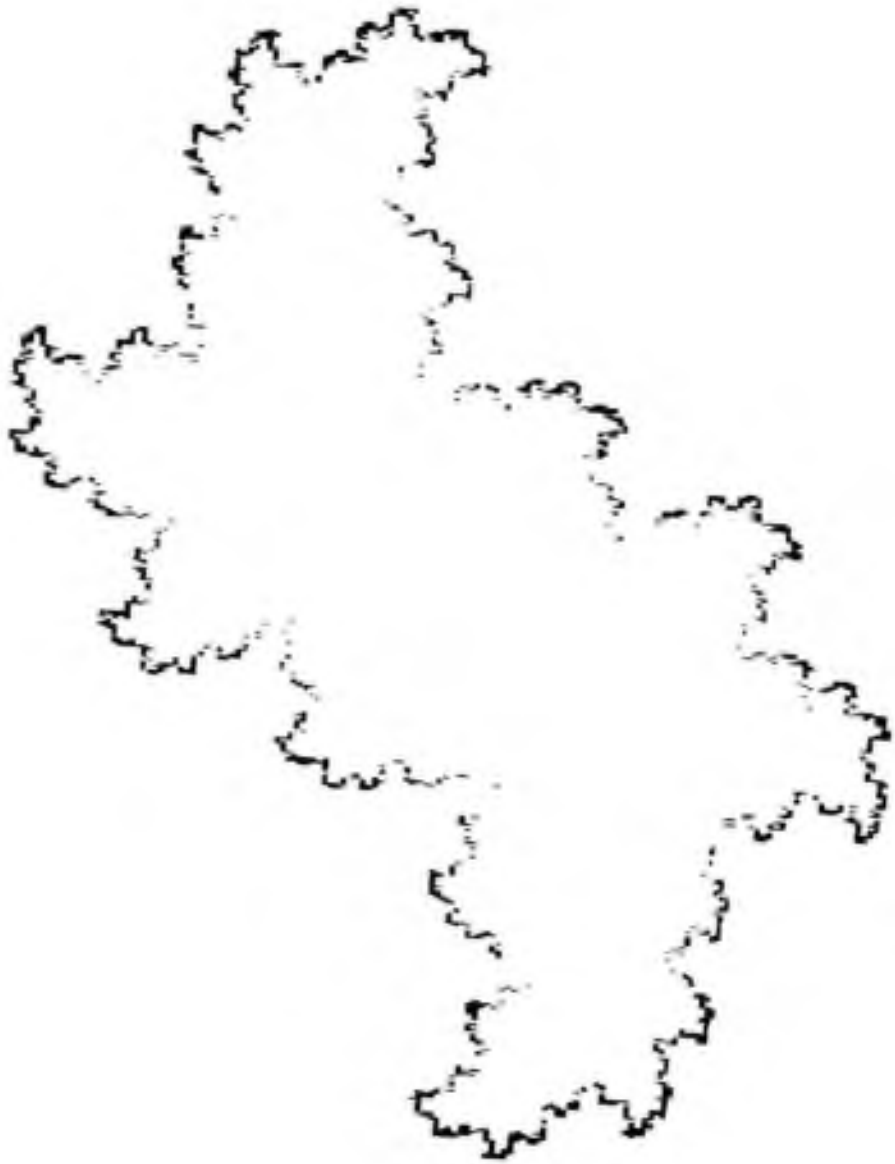


Fig. 14

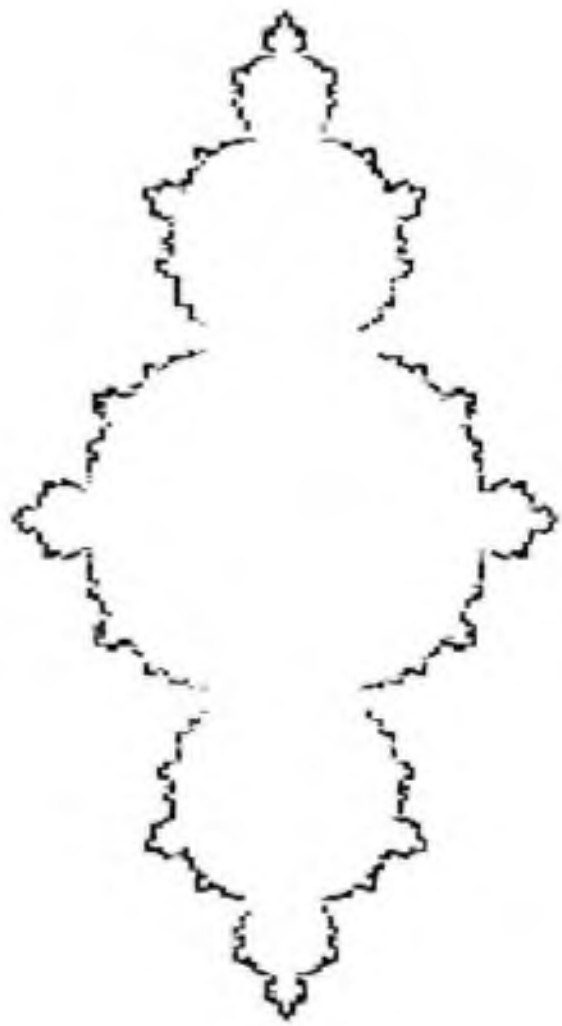


Fig. 15

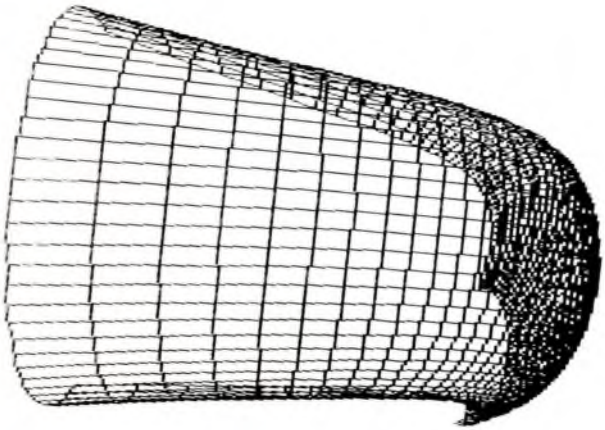


Fig. 16

En la siguiente tabla podemos ver los diferentes tiempos empleados para realizar las gráficas tanto en la máquina AT y en la monoputer.

FIGURA	TIEMPO REQUERIDO EN LA AT	EN LA MONOPUTER
Silla con 24x24 puntos calculados (Fig 11).	*3.5 minutos*	*3 segundos*
Membranas cuadradas (Figs. 12 y 13).	*2 segundos*	*0.05 segundos*
Curvas Fractales. (Figs. 14 y 15)	*2 minutos y 20 segundos*	*10 segundos*
Silla con 48x48 puntos calculados (Fig. 16).	*15 minutos y 23 segundos*	*11 segundos*

También se hicieron pruebas corriendo los mismos programas con otras dos máquinas una XT y una Vectra con coprocesador aritmético para tener más puntos de referencia. Los resultados de estas pruebas se dan en la siguiente tabla.

FIGURA	TIEMPO REQUERIDO EN LA XT	EN LA VECTRA
Silla con 24x24 puntos calculados (Fig 11).	*11 minutos y 54 segundos*	*29 segundos*
Membranas cuadradas (Figs. 12 y 13).	*8 segundos*	*1 segundo*
Curvas Fractales. (Figs. 14 y 15)	*4 minutos y 5 segundos*	*48 segundos*
Silla con 48x48 puntos calculados (Fig. 16).	*47 minutos y 5 segundos*	*1 minuto y 48 segundos*

Los tiempos requeridos para realizar cada una de las gráficas correspondientes a cada uno de los diferentes instantes, para la membrana son los mismos ya que se calcula el mismo número de puntos, solo que en diferentes instantes. Lo mismo ocurre con las dos curvas fractales mostradas, las cuales fueron generadas mediante la inversa de la función compleja $f(z)=\lambda(1-z)$ solo que para diferentes valores de λ.

El caso de la silla es diferente ya que para ésta se generó la gráfica apartir del mismo número de puntos pero en el primer caso se evaluaron 24x24 puntos y en el segundo 48x48 puntos, para obtener su superficie de aproximación. Hay que hacer notar que estos tiempos son sólo aproximados.

CONCLUSIONES

El presente trabajo nos ofrece una herramienta que nos permite emplear a la máquina monoputer 2 para la generación y despliegue de gráficas sin necesidad de emplear tarjetas especializadas, aprovechando las características tanto de la monoputer 2 como de la nodriza en que se encuentra.

El trabajo nos permite hacer gráficas directamente através de programas escritos en OCCAM y corriendo en la monoputer en forma indirecta por programas escritos en algun otro lenguaje corriendo en la nodriza.

Se proporciona una librería de funciones gráficas que ocultan los protocolos de comunicación al usuario, permitiéndole producir gráficas en forma sencilla, la cual puede ser modificada o aumentada por éste, si es que sus necesidades lo requieren.

Algo muy importante además del manejo de gráficas es que este trabajo deja abierto el camino para que con cierta facilidad se le hagan cambios o adaptaciones para aplicaciones especificas, las cuales no necesariamente tienen que ser sobre gráficas ya que también podemos manejar texto como se pudo notar.

BIBLIOGRAFIA

Sobre el lenguaje Occam2 y sus aplicaciones.

1.- Burns, Alan.
Programming In Occam 2
Addison-Wesley. 1988

2.- Jones, Geraint.
Programming In Occam 2
Prendice-Hall. 1988

Sobre la máquina monoputer 2.

3.- *Occam2 Standalone Compiler For the Transputer User's Manual*
Microway Inc. 1987

4.- *Monoputer 2, Owner's Manual*
Microway Inc. 1987

Sobre la máquina nodriza y sus sistema operativo.

5.- Duncan, Ray.
Advanced MS-DOS
Microsoft Press. 1986

Sobre el transputer.

6.- Jean-Daniel Nicoud
Andrew Martin Tyrrell
The Transputer T414 Instruction Set
IEEE MICRO, Junio de 1989.

7.- Stein, Richard M.
T800 *and Counting*
BYTE, Noviembre de 1989.

8.- Homewood, Mark
May, Dadiv
Shepherd, Dadiv
Shepherd, Rogers
The IMS T800 Transputer
IEEE MICRO, Octubre de 1987.

9.- Hull, M. Elizabeth
Zarea-Aliabadi, Adib
Real-Time System Implementation
The Transputer and Occam Alternative
Microprocessing and Microprogramming, Agosto de 1988.

Sobre los algoritmos empleados para la generación de gráficas.

10.- Newmman, William M.
Sproull, Robert F.
Principles Of Interactive Computer Graphics.
McGraw-Hill. 1984.

11.- Heran, Donald.
Baker, Pauline M.
Computer Graphics
Prentice-Hall. 1986.

12.- Rogers, David F.
Mathematical Elements for Computer Graphics
Mc Graw-Hill. 1976.

Sobre el lenguaje empleado para realizar los programas que corren en la máquina nodriza.

13.- Borland
Turbo C User's Guide
Borland, U.S.A., 1988.

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 22 de junio de 1990.

DR. JAN JANECEK

DR. ARMANDO MALDONADO TALAMANTES

DR. JAIME RANGEL MONDRAGON

AUTOR	CORTES GIL, EDUARDO
TITULO	INTERFAZ GRAFICA PARA LA MAQUINA MONOPITER 2
CLASIF.	XM 90.7
RGTRO.	BI-11731

NOMBRE DEL LECTOR	FECHA PREST.	FECHA DEVOL.
Lilia Olivares Vizquez	28-jul-97	8/08/97
Judith Esparza A.	8/sept/97	23/11/97
Edna Verónica Bonilla	11 nov 97	5 dic 97
Rodolfo Valdivia Ayala	08/oct/97	25/11/97
Patricia Castillo	13/oct/97	27/10/97
Dr. Guillermo Morales	5-12-97	