UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

UN EDITOR DE PANTALLA CON CODIGO COMPARTIBLE EN UN SISTEMA OPERATIVO NO IDEAL PARA EL COMPARTIMIENTO DE CODIGO

T E S I S

QUE PARA OBTENER EL TITULO DE:

MATEMATICO

P R E S E N T A :

SALVADOR LOPEZ MENDOZA

MEXICO, D. F. 1987.

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

INTRODUCCION

El Departamento de Matemáticas de la Facultad de Ciencias de la UNAM cuenta con una computadora PDP-11/34, esta computadora permite el uso simultáneo de sus recursos a varios usuarios bajo el sistema operativo RSX-11M. Al hacer un análisis del comportamiento del sistema operativo se notó que gran parte del tiempo se desperdiciaba en operaciones de lectura y escritura de programas a memoria secundaria, este problema se agrava al considerar que muchas de estas operaciones se efectuan al trabajar diferentes usuarios el mismo programa, con lo que se puede llegar al caso de que el sistema mande un programa a memoria secundaria para hacer lugar al mismo programa.

El propósito de este trabajo ha sido el de desarrollar un método para que aquellos programas que esten siendo utilizados simultáneamente no obliguen al sistema a efectuar demasiadas operaciones de lectura y escritura a disco. En particular en base a este método y considerando el tipo de trabajo que comunmente se desarrolla en esta computadora se desarrolló un nuevo editor de textos, que además permite un mejor ambiente de trabajo a los usuarios del sistema.

En el primer capítulo se describe la forma en que trabaja el sistema operativo y las consideraciones que llevan a proponer el desarrollo de programas compartibles como una forma de mejorar el rendimiento del sistema. Se describe también el esquema que se ha desarrollado para compartir código y un método para generar programas con esta característica.

En el segundo capítulo se propone el desarrollo de un editor de textos para ejemplificar el impacto de un programa compartible sobre el comportamiento del sistema operativo. Se esboza brevemente la historia de los sistemas de edición y se describe de manera general lo que es un sistema de edición.

En la tercera parte de este trabajo se describe la estructura del programa desarrollado, apegado al esquema propuesto para compartir código. Para ello se mencionan las características deseables en un buen editor de textos y los elementos considerados en la eleccción de aquellas que tiene el editor desarrollado. También se describen los problemas surgidos al dotar de algunas de sus características al editor y la forma en que se resolvieron.

DESCRIPCION DE LA COMPUTADORA PDP-11

La computadora es una herramienta que se ha desarrollado para ayudar al ser humano en su labor intelectual, de la misma forma en que las máquinas del siglo XVIII le ayudaron en su labor manual.

Estas herramientas serán más útiles en tanto que podamos aprovecharlas al máximo de sus capacidades. Este es el objetivo principal del presente trabajo, aunque sólo se considere una parte de lo que es un sistema de cómputo, su Unidad de Memoria. Antes de describir esta unidad es conveniente repasar someramente lo que es una computadora digital, a fin de tener un marco de referencia.

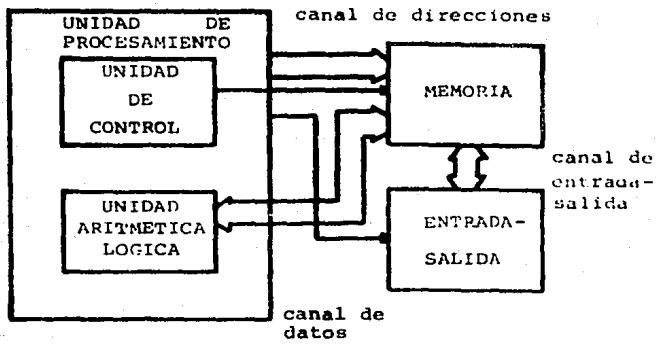
Se puede decir que una computadora se compone de las siguientes unidades:

UNIDAD DE PROCESAMIENTO. Es la encargada de efectuar las operaciones que le indiquen las instrucciones que componen el programa, para esto se sirve de las otras unidades, que son controladas por ésta. Para poder hacerlo cuenta con la Unidad de Control y la Unidad Aritmética y Lógica.

UNIDAD DE MEMORIA. En ésta se almacenan los programas que ejecuta la Unidad de Procesamiento, así como los datos que necesita y los resultados que genera.

UNIDAD DE ENTRADA Y SALIDA. Es la encargada de establecer la comunicación de la computadora con el exterior, esto se hace a través de diversos dispositivos (terminales, impresoras, discos magnéticos, cintas de papel y magnéticas, etcétera) .

Estas unidades interactúan entre sí, generalmente existen canales de comunicación por los que transitan los datos y señales de control, a dichos canales se les llama *BUS*. El diagrama siguiente muestra una de las posibles formas en que se relacionan dichas unidades.



Estructura de una computadora digital

Las computadoras PDP-11 tienen una estructura diferente a la mostrada anteriormente. Dicha estructura, que ha sido la base en el diseño de nuevos sistemas de cómputo, es la misma para todos los miembros de esta familia de computadoras, desde los sistemas dedicados a aplicaciones especificas hasta los grandes sistemas de tiempo compartido. A continuación se muestra el diagrama simplificado de la arquitectura de las computadoras PDP-11.

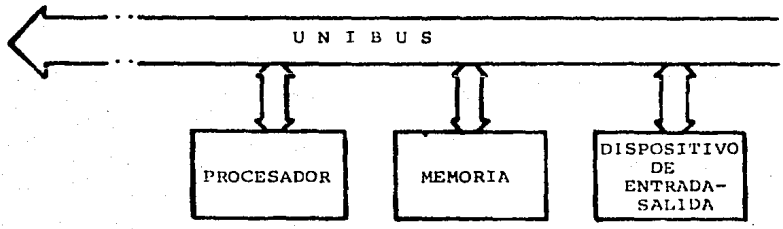


Diagrama de bloques del sistema PDP-11

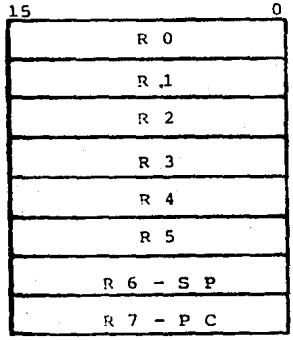
Como se ve todos los elementos se comunican entre sí por un único canal de comunicaciones (llamado UNIBUS). Todo elemento que esté comunicado a dicho canal, incluyendo localidades de memoria, los registros del procesador y los registros de los dispositivos periféricos, tiene asociada una dirección, que es usada para hacer referencia a dicho elemento. De esta forma el procesador puede manejar los registros de los dispositivos periféricos como si fueran localidades de memoria.

Las líneas que forman el canal de comunicaciones son bidireccionales y asíncronas, de manera que cada elemento del sistema puede enviar, recibir o intercambiar información sin intervención del procesador central.

Por otra parte, el hecho de usar un solo canal de comunicación para todos los elementos permite que sólo un elemento se comunique con otro en un momento determinado. Toda comunicación se efectúa estableciendo una relación AMO-ESCLAVO, pues uno de los dispositivos (el amo) toma control del canal de comunicaciones y maneja al otro (el esclavo) a través de las líneas de control.

Los elementos más importantes de este sistema son la unidad de procesamiento y la unidad de memoria, de manera que es conveniente tener una visión más detallada de su estructura y sus funciones.

PROCESADOR CENTRAL. Se encarga de efectuar tanto operaciones aritméticas y lógicas como la decodificación de instrucciones. Tiene un conjunto de 8 registros de alta velocidad que pueden usarse como acumuladores, registros de índice, apuntadores a memoria, apuntadores a una pila o aun para almacenar datos. De estos 8 registros (R0,..., R7) el R7 se utiliza como apuntador a la localidad de memoria que contiene la siguiente instrucción a ejecutar (contador del programa) y generalmente el R6 se usa como apuntador a una pila, señalando el último elemento insertado.



Registros del procesador

MEMORIA. La memoria puede verse como una serie de localidades con un número asignado a cada localidad (generalmente la primer localidad tiene el número 0 y así sucesivamente). Debido a que en la memoria de la PDP-11 se pueden alojar tanto bytes de 8 bits como palabras de 16 bits el número total de localidades de memoria no corresponde al número de palabras, sino al de bytes.

Dirección (Byte)			Dirección (palabra)
1	0	0	
3	2	2	
5	4	4	
7	6	6	
.	.	.	
.	.	.	
.	.	.	
777775	777774	777774	
777777	777776	777776	

Estructura de la Memoria

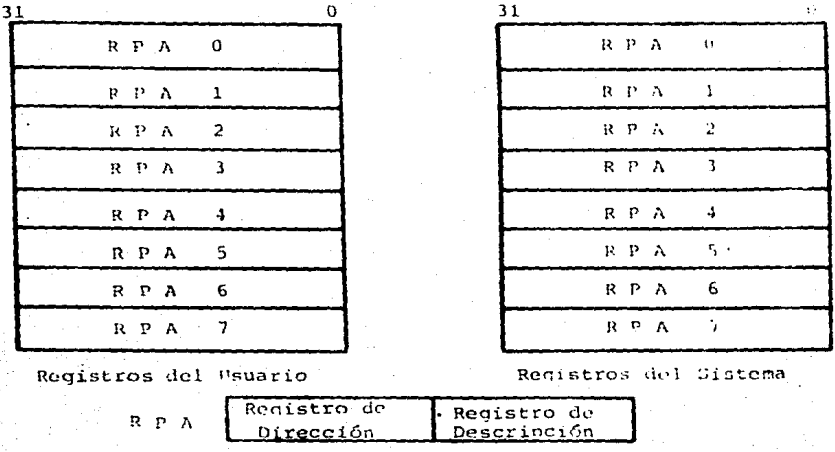
La familia de computadoras PDP-11 está diseñada para direccionar hasta 2^{16} (64 K) diferentes localidades de memoria, cada una de ellas de 8 bits o bien 2^{15} (32 K) palabras de 16 bits cada una.

En el caso de que se desee manejar un sistema de tiempo compartido este espacio de memoria resulta insuficiente, por lo que se diseñó un elemento que permite ampliar el espacio de memoria direccionable, a este elemento se le llama *Unidad de Manejo de Memoria* y con ella se puede direccionar un espacio de 2^{18} (256 K) bytes o 2^{17} (128 K) palabras. Sin embargo la presencia de la unidad de manejo de memoria es invisible para el usuario común, puesto que solamente utiliza 16 bits para hacer referencia a los elementos de memoria.

El hecho de tener una unidad de manejo de memoria permite, entre otras cosas, hacer uso eficiente del espacio total en sistemas de tiempo compartido y proporciona además los mecanismos de protección y relocalización necesarios en este tipo de sistemas.

Para entender con mayor claridad estos conceptos y la forma en que el sistema operativo RSX-11M hace uso de la unidad de manejo de memoria se describe con mayor detalle su funcionamiento.

La *Unidad de Manejo de Memoria* (UMM) tiene 2 conjuntos de 8 registros cada uno, llamados *Registros de Páginas Activas* (RPA). Uno de los conjuntos sólo es accesible a programas del sistema operativo y el otro es de uso general. En estos conjuntos se encuentra la información sobre el espacio de memoria que está utilizando el proceso actual. Cada uno de estos registros de 32 bits es en realidad una pareja de registros de 16 bits; un *Registro de Dirección de la Página* y un *Registro Descriptor de la Página*.



Registros de Páginas Activas

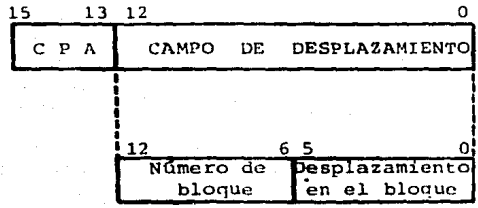
Estos conjuntos de registros se usan de la siguiente forma:

El espacio de memoria direccionable por cada proceso es de 32 K palabras, por lo que se necesitan 16 bits para hacer el direccionamiento. A este espacio accesible al proceso se le llama *Espacio Virtual*, dicho espacio se divide en 8 páginas que pueden tener hasta 4 K palabras cada una. Los 3 bits más significativos de la dirección se utilizan para obtener el RPA asociado a esa página, los 13 bits restantes se combinan con la información del registro de dirección de la página para formar una dirección en el *Espacio Físico* de la memoria.

Este mecanismo es el que permite separar el espacio virtual de un proceso en un conjunto de bloques en el espacio físico de la memoria, de manera que no se necesita de compactar la memoria para hacer un solo bloque de gran tamaño. Por otra parte este esquema permite fácilmente relocalizar programas, pues solamente es necesario cambiar el valor del registro de dirección de la página respectivo para que todos los direccionamientos se hagan a un bloque diferente de la memoria física.

El registro descriptor de la página tiene la información necesaria para controlar el acceso a la página correspondiente, de manera que un sistema operativo de tiempo compartido no tiene dificultad en establecer los mecanismos de protección necesarios para mantener la integridad de los diferentes procesos.

La palabra con que se especifica una dirección virtual se divide en 2 partes: el *Campo de la Página Activa* (CPA), que consta de 3 bits, y el *Campo de Desplazamiento* (CD), que ocupa los 13 bits restantes. Este último campo a su vez se divide en el *Número de Bloque* (NB) y el *Desplazamiento en el Bloque* (DB), ocupando 7 y 6 bits respectivamente.



Dirección Virtual de la Memoria

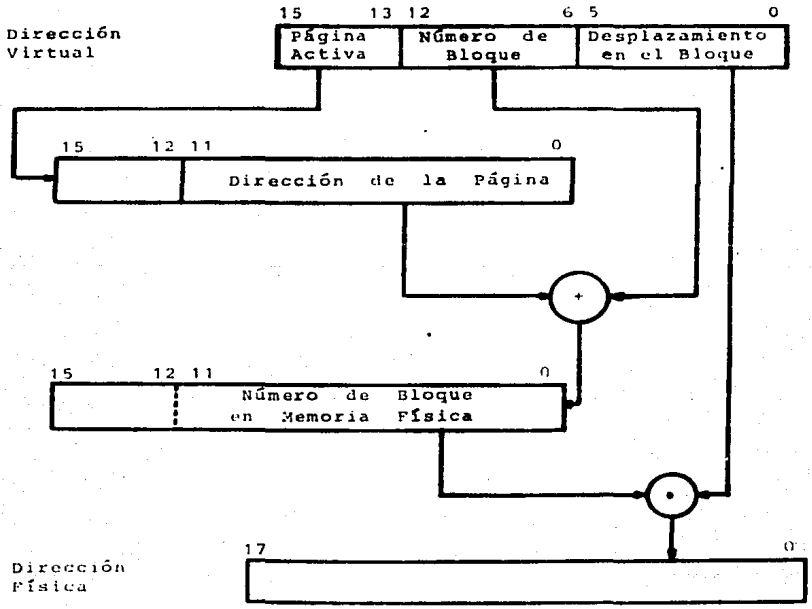
Dada una dirección virtual el CPA escoge una entre 8 posibles páginas (0–7) y el CD indica cúal de las 4 K palabras de esa página se desea manejar. A continuación se describen los pasos que sigue la unidad de manejo de memoria para crear una dirección física a partir de una dirección virtual:

Utiliza el campo de la página activa para seleccionar un registro de página activa (RPAn).

El campo de dirección de la página (12 bits) del RPA seleccionado se toma como la dirección inicial de la página actual. Esta dirección está dada como el número, en memoria física, del bloque inicial de la página.

Toma el número de bloque de la dirección virtual y lo suma al valor que tiene el campo de dirección de la página, obteniendo así el número del bloque en memoria física que contiene la dirección física deseada (12 bits).

Finalmente toma el valor correspondiente al desplazamiento dentro del bloque (6 bits) y lo concatena con el número de bloque obtenido en el paso anterior, de manera que se obtiene una dirección de 18 bits.



Construcción de una dirección física

MANEJO DE MEMORIA EN RSX-11M

El sistema operativo RSX-11M establece varios niveles de organización de la memoria. El primero de ellos es una división del espacio físico en un conjunto de *particiones* contiguas. El tamaño y las direcciones que ocupan estas particiones se definen al momento de configurar el sistema para una instalación determinada. Generalmente estas particiones son controladas por el sistema, pero existe la posibilidad de que un proceso pueda controlar una parte de alguna de ellas.

Dichas particiones pueden contener varias *regiones*, formando éstas un segundo nivel de organización de la memoria. Estas regiones pueden definirse como estáticas o como dinámicas, las primeras permanecen ocupando espacio de una partición hasta que alguna instrucción pida al sistema operativo que desaparezcan, en el caso de las regiones dinámicas el espacio de memoria sólo es utilizado mientras algún proceso esté trabajando con dicha región.

Los diferentes procesos se ejecutan en regiones, que pueden ser creadas automaticamente por el sistema en particiones controladas por él, o bien ser definidas y manejadas por el usuario en una partición controlada por algún proceso.

En caso de que algún proceso vaya a ejecutarse en una partición controlada por el sistema éste verifica que haya suficiente espacio contiguo para acomodarlo en memoria. Si el espacio es suficiente el código correspondiente al proceso se carga en dicha región y el proceso ya está listo para ejecutarse. En caso de que el espacio contiguo no sea suficiente para el proceso el sistema trata de hacerle lugar, ejecuta un algoritmo de compactación de la memoria si el tamaño total de los fragmentos libres es suficiente para que quepa el proceso (en este algoritmo se aprovechan las características de la UMM para relocalizar los procesos que mueve), si el espacio sigue siendo insuficiente busca aquellos procesos cuya prioridad sea menor que la del proceso demandante o aquellos procesos que esten suspendidos en espera de algún evento externo y manda una copia de dichos procesos, junto con la información sobre el estado del proceso, a alguno de los elementos de memoria secundaria. Una vez que hay espacio para el proceso éste se carga y puede ser ejecutado.

Es importante notar que al momento que el sistema trata de cargar un proceso para poder ejecutarlo no toma en cuenta la relación que pueda tener con los demás procesos, en particular se puede llegar al caso extremo en que un proceso tenga que ser enviado a memoria secundaria para hacer lugar a una copia del mismo proceso que haya sido activado por otro usuario, con

lo que el sistema estaría efectuando operaciones de escritura y de lectura en exceso, pues es suficiente con que se envie la información sobre el estado del proceso que ha sido suspendido y no el código.

El hecho de que diferentes usuarios esten haciendo uso del mismo programa simultaneamente también puede producir problemas, pues el sistema operativo puede colocar en memoria principal varias copias del mismo programa, que tengan como única diferencia los datos particulares de cada proceso.

Este esquema de manejo de memoria permite mantener varios procesos residentes en las diferentes particiones, en donde se ejecutan en forma concurrente. El sistema trata de colocar el mayor número de procesos en las particiones controladas por él, de manera que el procesador no tenga que esperar mucho tiempo para ejecutar algún proceso.

El espacio de memoria virtual asociado a cada proceso (32 K palabras) está dividido en 8 *ventanas*, cada una de ellas está asociada a una de las páginas en que se divide el espacio físico del proceso, el tamaño de estas ventanas varía en multiplos de 32 palabras, hasta un máximo de 4 K palabras (el tamaño de una página). Cada dirección virtual de 16 bits pasa por la unidad de manejo de memoria en donde es interpretada y mapeada a una zona del espacio físico.

En general estas ventanas y su manejo pasan inadvertidas para el usuario común, sin embargo el sistema operativo RSX-11M tiene entre las directivas accesibles a los usuarios un conjunto de 8 directivas para el manejo de memoria. Con estas directivas se pretende proporcionar al usuario las herramientas necesarias para permitirle aumentar el tamaño de su espacio virtual, sin embargo también pueden usarse para desarrollar algún tipo de manejo de memoria diferente al que tiene establecido el sistema operativo.

Estas directivas son:

CRAW - Crea una ventana virtual, permite definir sus características (tamaño, región en que se creará, lugar que ocupará en la región, etcétera).

CRRG - Crea una región dinámica en una partición controlada por el sistema. Especifica el tamaño de la región, el nombre que tendrá, la partición en que creará la región y los derechos de acceso para los demás procesos.

ATRG - Ata una región al proceso demandante, la región puede ser estática o dinámica (es necesario atar al proceso aquellas regiones que quiera usar).

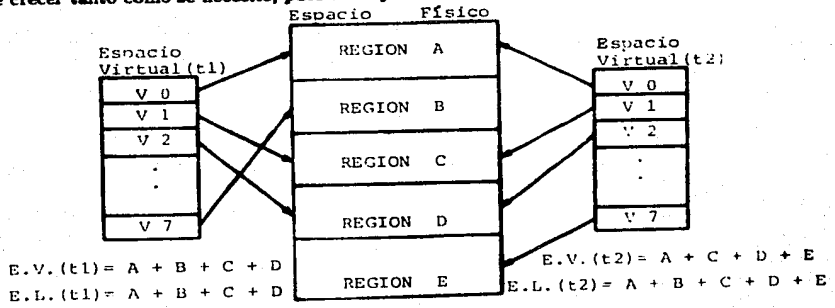
MAP - Mapea una ventana a una región. Toma los atributos dados a la región y la ventana al momento de crearlas.

DTRG - Libera al proceso de la región especificada, automáticamente elimina el mapeo de las ventanas que aún estén mapeadas.

UMAP - Elimina el mapeo existente entre la ventana especificada y alguna región.

ELAW - Elimina una ventana existente. En caso de que aún estuviera mapeada a alguna región primero elimina el mapeo.

El uso de estas directivas de manejo de memoria permite tener programas que utilicen más de 32 K palabras de memoria, pues un proceso puede mapear sus ventanas a ciertas regiones inicialmente y en el transcurso de su ejecución mapear alguna de esas ventanas a otra región. Sin embargo en todo momento el espacio virtual que se maneja tiene a lo más 32 K palabras. Al espacio total que puede tener un proceso durante su ejecución se le llama *Espacio Lógico* y puede crecer tanto como se necesite, pues no hay límite a su tamaño.

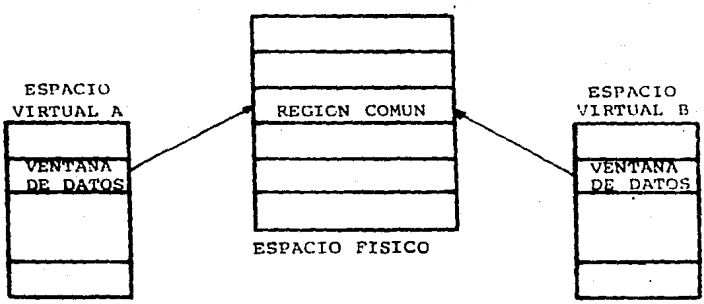


Espacio Lógico de Direccionamiento

ESQUEMA PARA COMPARTIR CODIGO

Las directivas para manejo de memoria pueden usarse para fines diferentes al de aumentar el espacio lógico de los programas, en particular existe la posibilidad de tener varios procesos mapeando una o más de sus ventanas a una región común al mismo tiempo.

Para lograr este propósito alguno de los procesos tiene que crear una región, usando la directiva CRRG, definiendo también las restricciones de acceso a dicha región. Una vez que se tiene creada la región todos aquellos procesos que quieran mapearse a ella podrán hacerlo, primero atando la región al proceso y después creando y mapeando una ventana a dicha región.



Mapeo de Ventanas Virtuales a Memoria Física

Es importante notar que las ventanas de cada proceso que se mapean a la región pueden ser diferentes, de manera que la región común no ocupe el mismo espacio virtual para los diferentes procesos. También es importante darse cuenta de que los diferentes procesos no están restringidos a pertenecer a una partición en particular, puede darse el caso de que tanto los procesos como la región común estén en la misma partición.

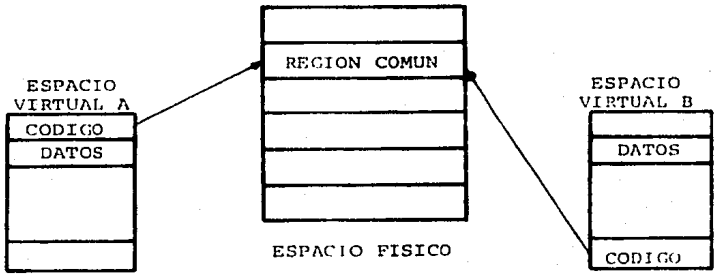
Esta forma de compartir regiones permite establecer un mecanismo para manejar concurrentemente un área de datos común. En este caso la situación puede presentarse como se muestra en la figura anterior, puesto que las referencias que haga el código a los elementos de esta región son en términos de direcciones virtuales, de manera que la unidad de manejo de memoria es la encargada de hacer el direccionamiento físico correcto.

En caso de que la región común contenga código éste deberá ser *reentrante*, es decir no deberá tener ningún tipo de datos ni debe modificarse durante su ejecución. El hecho de que el código sea reentrante garantiza que puede ser suspendido en cualquier punto durante su ejecución como parte de algún proceso y que ese mismo código pueda ser usado en otro proceso, al final del cual puede continuar con la ejecución del proceso suspendido sin afectarlo, de hecho para el proceso suspendido puede pasar inadvertida toda esta situación.

Otro aspecto que debe considerarse al escribir rutinas reentrantes es el manejo adecuado de las referencias que hagan dichas rutinas a las localidades de memoria. Se pueden mencionar dos formas de resolver este problema en el sistema operativo RSX-11M.

1. **Utilizando apuntadores a los datos, dichos apuntadores se pueden pasar a la rutina reentrante en los registros del procesador. De esta forma se aprovecha el hecho de que cada proceso tiene asociado un conjunto de valores para sus registros y estos son almacenados cada vez que el proceso es suspendido. Al manejar apuntadores todas las referencias a alguna localidad del espacio virtual se hacen de manera indirecta, esto permite tener mayor independencia de la rutina con respecto a la zona en que se encuentran los datos.**

2. **Haciendo referencia a elementos en una zona del espacio virtual que sea propia de cada proceso. De esta forma la zona de datos debe estar en las mismas direcciones del espacio virtual de los diferentes procesos. En este caso el código compartido puede encontrarse en zonas diferentes del espacio virtual de cada proceso, siempre y cuando, además de ser reentrante sea ejecutable independiente de la posición en que se encuentre. La figura muestra un ejemplo de esta última situación.**



Procesos Compartiendo una Región de Código

En base a lo mencionado en la sección anterior se puede esquematizar la estructura de programas con código compartido como sigue:

Por una parte se define el módulo o módulos que contienen a las rutinas reentrantes y por otra parte los módulos que serán particulares a cada usuario, en estos últimos debe reservarse una zona de la memoria para datos y otra con código que en particular debe hacer referencia al código común.

Una vez que se han construido estos módulos, se debe establecer un mecanismo para colocar el código de la rutinas reentrantes en una región particular de la memoria, de tal forma que esté disponible para cuando algún proceso lo necesite.

El sistema operativo RSX-11M permite manejar rutinas con código compartido definiendo *bibliotecas residentes*, éstas se construyen utilizando el *Constructor de Tareas* (TKB), que genera un archivo con la imagen del código ejecutable y otro con una tabla que contiene la información sobre la región común. Ya que se tiene el archivo con la imagen del código se reserva una región (estática) de la memoria y en ella se carga el código.

Cuando se construye la imagen correspondiente al programa que emplea las rutinas comunes se debe especificar la tabla con la información sobre las rutinas comunes. Al activar este programa las referencias al código compartido ya están resueltas y por la forma en que se construyen las bibliotecas residentes dicho código ya está en memoria, de tal forma que el proceso de cada usuario puede completar su ejecución sin dificultad.

```
>MAC RUTCOM = RUTCOM
>TKB
TKB>LB:[1,1]RUTCOM/-HD/MM,,LB:[1,1]RUTCOM=RUTCOM
TKB>/
TKB>STACK=0
TKB>PAR=COMUN:160000:1000
TKB>//
>SET /MAIN:COMUN:1400:100:COM
>INS LB:[1,1]RUTCOM/PAR=COMUN
>MAC PROG=PROG
>TKB
TKB>PROG=PROG
TKB>/
TKB>LIBR=RUTCOM:RO
TKB>//
```

Construcción de una Biblioteca Residente

Es importante notar que para construir programas con código compartible siguiendo este esquema no se utilizan explícitamente las directivas de manejo de memoria, lo que facilita su desarrollo. Sin embargo, existe un gran problema con este método, la región que contiene el código compartible es estática, es decir ocupa lugar en memoria sin importar si el código que contiene está siendo usado o no. De hecho para crear la región, cargarla con el código y eliminarla cuando ya no se use, se necesita la intervención de un operador. El hecho de que la región sea estática se contrapone al objetivo que se persigue al utilizar código compartido: hacer mejor uso de la memoria, pues habrá momentos en que la región compartible estará ocupando espacio en la memoria física sin que algún proceso la necesite, desperdiciando una zona de la memoria que podría ser usada por otros procesos.

Como alternativa a este método se desarrolló un esquema diferente para construir programas con código compartido, dicho esquema presenta mayores dificultades para escribir los programas, pues se utilizan las directivas de manejo de memoria para establecer la correspondencia entre el código compartible y los diferentes procesos que lo necesiten. A continuación se describen los elementos que se tienen que considerar al construir los módulos para apegarse a este nuevo mecanismo.

Al escribir el módulo que contiene las rutinas reentrantes se debe considerar que los datos ocuparán alguna zona del espacio virtual. Esta zona debe ser respetada por todos los programas que quieran usar las rutinas compartibles, de hecho para las rutinas compartibles la zona de datos pertenece a su espacio virtual. Un factor que influyó notablemente en la decisión de conservar esta zona del espacio virtual para los datos fue la velocidad de respuesta del programa desarrollado, en general es más rápido un direccionamiento absoluto que uno indirecto. Esta restricción puede ser eliminada en otro tipo de programas, sobre todo aquellos que no tengan que establecer diálogos muy prolongados con el usuario.

También se debe escribir un cargador para el código reentrante, este cargador solo debe ser activado en caso de que aún no exista la región con el código compartible, así que debe crear la región común, mapear una ventana a ella y copiar el código de las rutinas en dicha zona. Este programa trabaja en sincronía con el programa que utiliza las rutinas compartibles, el cual permanece suspendido mientras el cargador hace su trabajo, una vez que la región común tiene el código compartible el primer programa puede continuar su ejecución.

El programa que invocará el usuario sólo debe tener el área de datos (que será exclusiva del usuario en cuestión) y el código necesario para establecer una ventana a la región con el código

compartible. Como se ha visto este programa puede pedir al sistema operativo la activación del cargador en caso de que aún no exista la región común. Para construir estos dos programas se utiliza el TKB de la siguiente forma:

Se construye una imagen ejecutable para el programa que invocará cada usuario, al hacerlo se debe generar una tabla con las direcciones virtuales que hayan sido asociadas a las distintas variables del área de datos. Utilizando esta tabla se escribe un módulo en el que se definen estas mismas variables como un conjunto de símbolos con los valores ya determinados. Este último módulo se ensambla para después ligarse con las rutinas compartibles, de manera que estas ya puedan resolver las referencias a las variables particulares de cada usuario.

Esta forma de resolver las referencias que hacen las rutinas compartibles a variables del área particular a cada proceso no es la más simple, pues se puede definir dentro del módulo que contiene a estas rutinas un área igual (tanto en su tamaño como en su posición en la memoria virtual), al área de datos del programa que utilice dichas rutinas, sin embargo usando esta última forma el tamaño del cargador aumenta por un área que no se usará, así que en un afán de reducir el tamaño de los programas se optó por el método ya descrito.

De esta forma cuando algún usuario active al proceso que usará el código compartible la región común será creada en caso de que aún no exista, y cada nueva invocación de dicho programa sólo incrementará el espacio de memoria utilizada en el tamaño que ocupe cada área de datos. Una vez que todos los procesos que compartan esta región común terminen su trabajo el sistema operativo se encargará de eliminarla de la memoria principal, liberando así todo el espacio que se haya usado.

Como se ve este proceso es más elaborado que el propuesto por los diseñadores del sistema operativo, sin embargo la ganancia que se obtiene en cuanto a memoria disponible justifica el trabajo adicional.

REFERENCIAS

Para una descripción más completa de lo que es un sistema de cómputo se puede consultar cualquier libro de *Organización de Computadoras*, de estos se pueden recomendar [HAM-84], [STO-75]. Respecto al caso particular (PDP-11) la mejor referencia es el manual del procesador [DEC-76]; en algunos libros se describe esta familia de computadoras de una forma más simple, por ejemplo [STO-75], [MAC-81], [WAK-81], este último hace una breve mención a lo que es código reentrante. Una descripción más completa de las características que debe tener el código para ser reentrante se encuentra en [GRI-78].

La información concerniente al manejo de memoria, en el aspecto físico, está tomada principalmente del manual del procesador y algunos conceptos de [WAK-81].

Una buena referencia para estudiar lo que es un sistema operativo y la forma en se administra la memoria se puede encontrar en [CAL-82], [HSI-75], [LIS-81] y [LOR-81].

En cuanto al funcionamiento del sistema operativo RSX-11M, la información más completa está en los diferentes manuales del sistema. Una descripción general está en [DEC-77a], la descripción de las directivas al sistema operativo se encuentra en [DEC-77b] (ahí también se hace mención a lo que es manejo de memoria en este sistema operativo), finalmente todo lo relativo a las bibliotecas residentes se encuentra en [DEC-77c].

EDICION DE TEXTOS

Un editor es un programa de computadora que permite crear, revisar y modificar documentos, entendiendo por documento cualquier cosa que pudiera encontrarse en una página impresa. Cuando el tipo de documento que maneja el editor está formado por cadenas de caracteres, se tiene un *Editor de Textos*, con él se pueden producir tanto programas para computadora como textos simples. Existen otros tipos especiales de editores, como los *Editores Gráficos*, que permiten crear y modificar gráficas (en algunos casos se manejan editores gráficos para aplicaciones muy particulares, como el diseño de circuitos electrónicos). En lo que resta de este trabajo se consideran unicamente editores de textos.

El proceso de edición de textos actualmente es muy simple para el usuario, tanto que se ha llegado al punto de tener sistemas especializados hacia el desarrollo de programas en algún lenguaje particular (editores guiados por la sintaxis) y hacia la elaboración sofisticada de textos (procesadores de palabra), sin embargo la edición de textos no siempre ha sido tan simple como lo es actualmente. Para comprender con mayor claridad la situación imperante al inicio de este trabajo es conveniente revisar, aunque sea someramente, el desarrollo de los editores. Una vez establecido el marco de referencia se muestra un modelo para describir un *sistema de edición*.

BREVE RESEÑA HISTORICA

En los principios de la computación no existía el trabajo interactivo entre el usuario y la máquina, de hecho para programar una computadora era necesario conocer muy a fondo la estructura interna de la misma, pues la secuencia de instrucciones se establecía alambrando las tarjetas de control; como se ve el uso de las computadoras estaba reservado a un grupo muy reducido de personas. Además de ser un proceso muy tedioso también era un proceso muy vulnerable a los errores, lo que hacía muy lento el desarrollo de los programas. Por otra parte todos los recursos del sistema estaban asignados al proceso que se estuviera ejecutando, aunque dicho proceso no tuviera necesidad de algunos de ellos. Estas situaciones forzaron la creación de los *sistemas operativos*, de manera que los recursos de la computadora fueron usados de una forma más racional; al mismo tiempo se simplificaron los mecanismos de acceso a los recursos de la computadora.

Los primeros sistemas operativos utilizaron las tarjetas perforadas como el medio en que el usuario podía escribir su programa de manera tal que la computadora también pudiera entenderlo. De esta forma el proceso para crear y ejecutar un programa se hacía en dos partes:

1. La elaboración del conjunto de tarjetas que componían el programa se efectuaba sin ningún contacto con la computadora, utilizando máquinas perforadoras de tarjetas. El usuario era el responsable de ordenar adecuadamente sus tarjetas y en caso de que hubiera algún error debía localizarlo y corregirlo.

2. Una vez elaborado y organizado el conjunto de tarjetas era "alimentado" a la computadora, esto se hacía utilizando una lectora de tarjetas perforadas. Dicho conjunto de tarjetas era tratado por la computadora como se lo indicase un subconjunto de las mismas tarjetas, las *tarjetas de control*; en caso de que se tratase de un programa en las tarjetas de control se indicaba a la computadora que debía de compilarlo y, en caso de no haber errores en dicho proceso, ejecutarlo tomando como datos otro subconjunto de tarjetas.

Este proceso ya era más simple que el anterior y permitió el desarrollo de programas en menor tiempo, sin embargo presentaba serias dificultades; por ejemplo cualquier corrección, por pequeña que fuera, obligaba a alimentar de nuevo todo el conjunto de tarjetas y en caso de que la corrección fuera sobre lo escrito en alguna de las tarjetas ésta debía ser eliminada y reemplazada por otra con el texto ya corregido; si se tenía que hacer el mismo cambio en varias tarjetas era responsabilidad del programador localizar cada una de las tarjetas involucradas y hacer la modificación correspondiente.

Algunos de estos problemas se resolvieron con perforadoras de tarjetas más sofisticadas, por ejemplo eran capaces de mantener en memoria los caracteres tecleados y esperar una indicación para proceder a perforar la tarjeta, de esta forma existía la posibilidad de corregir alguna parte del texto antes de perforar la tarjeta; también era posible duplicar una tarjeta, de manera que reemplazar una palabra por otra no obligaba al programador a reescribir toda la tarjeta. Es importante notar que esta facilidad sólo era práctica si las palabras en cuestion tenían el mismo número de caracteres.

A pesar de estas inovaciones la edición de programas seguía siendo un proceso muy tedioso, para resolver algunos de los problemas que aún subsistían se crearon *editores de tarjetas*. Estos eran programas que esperaban como datos dos conjuntos de tarjetas, uno con el programa a editar y otro con las modificaciones que debía hacer al primer conjunto. Para poder hacer estas tareas el editor asociaba (implícitamente) a cada tarjeta del primer conjunto de datos un número de línea, de esta forma las operaciones a efectuar se especificaban por parejas (número de tarjeta y operación a efectuar con dicha tarjeta). Con estos editores no se podían volver a corregir aquellas porciones del texto que ya hubieran pasado, por lo que la secuencia de

instrucciones estaba ordenada por el número de línea.

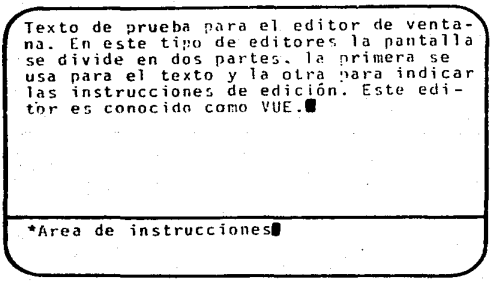
Con la evolución de los sistemas operativos también se dió una evolución en los ambientes de trabajo, así con los sistemas operativos de *tiempo compartido* ya aparecieron programas *editores interactivos*. Estos editores se trabajaban generalmente en terminales con impresión en papel y la unidad de trabajo era una línea de texto. Los primeros editores se basaban en el mismo modelo conceptual del trabajo con tarjetas perforadas, generalmente asociaban un número a cada línea (en la mayoría de estos editores dicho número estaba implícito pero en otros se mostraba explicitamente) y las instrucciones de edición seguían el mismo formato que los editores de tarjetas. De ahí que a dichos editores se les llamara *editores de línea*.

Con estos editores ya fue posible hacer operaciones especificando un contexto, de manera que el editor primero localizaba el texto en cuestión y después hacía la edición; además ya fue posible efectuar operaciones de edición sobre porciones del texto que estuvieran antes de la línea vigente en ese momento, aunque en la mayoría de los casos el movimiento estaba limitado a sólo unas líneas. Esto era debido a restricciones en la memoria, pues como no era suficiente para tener todo el texto presente se tenía que dividir en bloques y sólo una porción se encontraba disponible, de esta forma existía completa libertad para moverse sobre el texto del bloque actual, sin embargo no era posible moverse libremente de un bloque a otro, generalmente se estaba restringido a moverse unicamente hacia adelante y para poder regresar había que terminar la edición y volver a empezar. Esta restricción se debía principalmente a que los sistemas de manejo de archivos unicamente permitían accesos muy eficientes sobre archivos secuenciales. Entre estos editores destacan *EDI* (en sistemas desarrollados por DEC), *ED* (en UNIX) y *CANDE* (en las computadoras B6700).

A pesar de las ventajas que se obtuvieron en estos editores aún existían limitaciones importantes, el hecho de trabajar con líneas como unidad restringía el tipo de instrucciones posibles, de hecho operaciones más sofisticadas como localizar cadenas de caracteres que no estuvieran en la misma línea eran imposibles. Otra limitante importante era el tamaño de la línea de trabajo, cuyo tamaño máximo generalmente era de 80 caracteres (restricción heredada de los editores de tarjetas perforadas). Un aspecto que también puede considerarse como restrictivo era el conjunto de caracteres manejables, pues muchos de estos editores sólo permitían manejar letras mayúsculas (esta restricción estaba determinada por el tipo de terminales existentes, que a su vez lo habían heredado de las máquinas perforadoras de tarjetas). Con todo y estas restricciones los editores de línea ya fueron empleados para la elaboración de textos por computadora, sin embargo los problemas que presentaban desalentaron a muchos posibles usuarios.

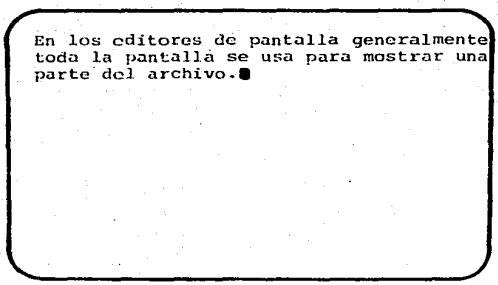
Algunos de los problemas de los editores de línea fueron superados con la aparición de los *editores de caracteres*, en ellos todo el texto se ve como una gran cadena de caracteres (en la mayoría de los casos la representación visual se muestra como un conjunto de líneas pero internamente no existen fronteras). Entre estos editores destaca *TECO*, pues no sólo se puede editar interactivamente con él sino que se pueden elaborar todo tipo de programas que hagan manipulación de texto (ordenamientos, busqueda de palabras, etcétera). Esta característica ha permitido que TECO sea usado como núcleo de editores más sofisticados.

Con la aparición de las terminales de despliegue en pantalla de video se crearon otro tipo de editores, pues si bien es cierto que todo lo que se hacía en una terminal con impresión en papel se podía hacer en una terminal de video no se aprovechaban las capacidades de estas últimas. Los primeros editores que manipularon la pantalla de las terminales de video mostraban una sección del documento en la pantalla pero sólo se podía editar una parte de ella, generalmente una línea.



Editor de Ventana

La mayoría de este tipo de editores eran en realidad extensiones a los editores existentes (en general editores de línea), a estos editores se les llamó *editores de ventana*, como un ejemplo se tiene a VUE, que es una extensión de TECO para terminales VT-100. Este editor divide la pantalla en 2 zonas, en una de ellas se tiene una parte del texto (20 líneas) con un cursor que indica en que parte de él se está trabajando y en la otra aparecen las instrucciones de edición conforme el usuario vaya escribiendolas.



Editor de Pantalla

Aunque este ambiente de trabajo era mejor que el existente anteriormente aún no se explotaban las capacidades de las terminales de video. En 1972 se propusó un modelo conceptual de las características que debía tener un *editor de pantalla*, entre estas destacan:

El texto se considera como una página que se extiende indefinidamente tanto a lo largo como a lo ancho y el caracter del extremo superior izquierdo está en el origen del archivo. Existe una señal visual que indica en que porción del texto se está trabajando, a esta señal se le llama *cursor*.

El usuario puede desplazarse sobre el documento usando teclas especiales para indicar sus movimientos, generalmente estos movimientos son relativos a la posición actual del cursor.

Las modificaciones al texto se pueden hacer colocando el cursor sobre el caracter deseado y reescribiendo sobre él.

La inserción de nuevo texto se hace en la posición que señala el cursor.

En cualquier momento el usuario ve una versión actualizada y precisa de la porción del documento que quepa en la pantalla.

Generalmente en este tipo de editores las instrucciones de edición se invocan usando teclas de control, pues todo aquello que sea imprimible es considerado texto a insertar.

Este tipo de editor es el que se encuentra regularmente en toda instalación; la mayoría son versiones sofisticadas del modelo descrito, pues permiten manipular bloques de texto (palabras, oraciones, párrafos), recuperar texto ya borrado y algunos incluyen instrucciones que facilitan la programación en algún lenguaje en particular (por ejemplo en Z y en EMACS se pueden definir parejas de delimitadores y usarlos en instrucciones especiales, de manera que al pedirle que encuentra la pareja de la palabra BEGIN el cursor se posicione sobre la palabra END correspondiente). En la mayoría de estos editores ya no existen restricciones en cuanto al manejo del texto, pues siempre está disponible todo; la manipulación del archivo con el texto es implícita, lo que libera al usuario de las limitaciones de otro tipo de editores. Estos editores son los más conocidos actualmente y entre ellos podemos destacar a *EMACS* (cuya versión reducida para microcomputadoras se llama *MINCE* o *PERFECT WRITER*), *Z*, *VI* y el editor de *TURBO PASCAL*.

Por otra parte se desarrollaron editores que manipulan documentos en términos de su *estructura*, el primero de ellos, *NLS*, podía manejar textos en base a su jerarquía, este tipo de editores fueron la base para desarrollos en otras áreas pues a principios de los años 70 se extendió este concepto hacia los *editores guiados por la sintaxis*, en los que se impone la estructura de un lenguaje de programación. Estos editores tratan de asegurar que el programa en desarrollo se apegue a la sintaxis del lenguaje, además en algunos de estos sistemas a la vez que se edita el programa se genera una representación intermedia para que el compilador trabaje sobre ella en las etapas restantes del proceso de compilación (análisis semántico y generación de código). Este tipo de editor ha tenido un gran auge a últimas fechas, pues actualmente se les trata de integrar a ambientes de desarrollo de programas. Entre los más conocidos de estos proyectos se encuentran el *Sintetizador de programas de Cornell* y el sistema *Interlisp*.

También usando una estructura particular se han desarrollado *procesadores de palabras* interactivos, estos programas permiten manipular facilmente documentos efectuando operaciones muy particulares: centrado de texto, alineación con respecto a los margenes, indentación de párrafos, numeración de páginas y en algunos casos se tiene silabeo automático (si una palabra ya no cabe en la línea actual se parte de acuerdo a las reglas sintácticas de la grámatica); si esta última característica no se tiene, generalmente se proporciona la de no truncar aquellas palabras que excedan los margenes de la hoja, dichas palabras se acomodan en las líneas siguientes y además se distribuye el espacio libre proporcionalmente entre las palabras restantes

de la línea original. Uno de los procesadores de palabras más conocido es *WORD STAR*, que también puede usarse para editar programas.

Con la aparición de terminales más sofisticadas han aparecido *editores gráficos*, que permiten el manejo de elementos que anteriormente no se usaban. Este tipo de sistemas originalmente se usaba para el *diseño auxiliado por computadora* y con el gran desarrollo que han tenido las computadoras personales, ha sido posible crear sistemas de desarrollo de programas en los que se combinan la manipulación de elementos gráficos con la edición, de tal forma que es muy sencilla la interacción con ellos. Estos sistemas se usan también en la composición de documentos cuyo texto incluye simbología diferente a la proporcionada por un teclado simple (un buen ejemplo de este tipo de documentos es el texto de las publicaciones matemáticas). En estos sistemas el usuario dispone ya de una representación (en la pantalla) de la forma en que quedará su documento. Como ejemplos importantes de estos sistemas está *SMALLTALK-80* como sistema de desarrollo y *WORD* representativo de los procesadores de palabras sofisticados.

MODELO DEL SISTEMA DE EDICION

A lo largo de este trabajo se ha tratado lo que funcionalmente se considera como un editor de textos interactivo, por lo que es conveniente formalizar estos conceptos.

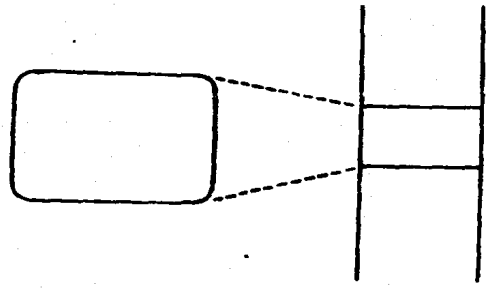
Todo sistema de edición puede describirse en dos formas: desde el punto de vista del usuario y desde el punto de vista del diseñador y realizador del sistema.

Para el usuario el sistema tiene dos componentes fundamentales: Un *Modelo Conceptual* del sistema y la *Interfaz* con que trabaja.

El modelo conceptual describe el ambiente en que se puede trabajar, proporciona una abstracción del documento y sus elementos, y permite al usuario anticipar el efecto de las operaciones que ordene sobre el documento. De lo anterior se desprende la importancia de un buen modelo conceptual, sin embargo en muchos sistemas de edición dicho modelo no está bien especificado, ya sea porque es poco visible o porque no es consistente. En estos casos el uso del sistema es poco satisfactorio, pues quien interactúe con él no podrá establecer principios que guíen su trabajo. El modelo conceptual no sólo es de ayuda para el usuario del sistema, pues también puede ser usado como un auxiliar por alguna persona que pretenda desarrollar su propia versión de dicho sistema, por lo que una buena especificación le será muy útil.

Como ejemplo de modelo conceptual se puede mencionar el de los primeros editores de línea, que pretendían simular el "mundo" de las tarjetas perforadas, de esta forma el usuario sólo podía ver la línea (tarjeta) en que se trabajaba y las operaciones que efectuaba se aplicaban sobre dicha línea.

Otro ejemplo interesante y de más actualidad lo constituyen los editores de pantalla, en que se piensa el documento como una hoja con un conjunto de líneas. Esta hoja es de tamaño infinito, tanto a lo largo como a lo ancho, en este caso se tiene a la vista un subconjunto del texto total (generalmente lo que cabe en la pantalla), es como tener una rendija por la que se puede observar el texto, con la posibilidad de mover dicha ventana al punto que se desee; bajo este modelo las operaciones son sobre porciones del texto sin importar los límites de líneas o páginas.



Modelo conceptual de editor de pantalla

Generalmente los futuros usuarios del sistema sugieren a los diseñadores las características que debe tener, de dichas sugerencias surge el modelo conceptual. Sin embargo cada usuario establece para sí un modelo del sistema; que es la forma personal en que entiende al modelo conceptual, los elementos del sistema y la forma en que se relacionan, así como las operaciones que puede aplicar a dichos elementos y la forma de invocarlas. Este modelo es conocido como *Modelo del Usuario* y generalmente se forma a través de las experiencias que deja el interactuar con el sistema, aunque también influye la información obtenida en la documentación existente (manuales) o por consejos de algún otro usuario.

El modelo del usuario no siempre concuerda con el modelo conceptual del sistema, cuando el usuario sólo usa algunas de las operaciones posibles o cuando las aplica limitadas a operar sólo sobre ciertos elementos se puede pensar que su modelo es un subconjunto del modelo conceptual; en cambio si usa las operaciones existentes para crear otras u otra forma de aplicarlas, definiendo así operaciones no contempladas originalmente en el modelo conceptual, se puede decir que su modelo es una extensión del modelo conceptual; finalmente en algunas ocasiones el modelo del usuario es equivalente al modelo conceptual, aunque esta equivalencia sea operativa puede ser que lógicamente sean diferentes, es el caso en que un editor de pantalla es visto por el usuario como un documento que se mueve hacia cualquier lado, dejando ver una porción del texto a través de una ventana fija, y no como se describió antes.

El otro elemento que forma parte del sistema es la interfaz con que trabaja el usuario, ésta es el conjunto de elementos de que dispone el usuario para comunicarse con el sistema de edición. Los elementos de que consta son tres: Los dispositivos de entrada, los dispositivos de salida y el lenguaje de interacción.

Los dispositivos de entrada permiten al usuario añadir elementos al documento, indicar al sistema la operación a efectuar y marcar los elementos editables. En la mayoría de las ocasiones sólo se tiene un dispositivo de entrada, el teclado de la terminal, que en general es del tipo conocido como QWERTY (por las 6 primeras letras del primer renglón de teclas); aunque ha habido intentos por cambiar la disposición de las teclas este tipo de teclado ha adquirido ya una presencia muy difícil de superar, sin embargo su diseño inicial (que correspondía al teclado de una máquina de escribir) ha sido ampliado con teclas de uso especial: un teclado numérico especial, teclas para indicar movimientos en la pantalla y teclas para indicar funciones especiales.

Los teclados caen dentro de la categoria de *dispositivos de texto*. Otra clase de dispositivos son los de *elección* o de *botón*, en los que al oprimir un botón especial se invoca alguna acción (en algunos casos estos botones son parte de la extensión que se hace al teclado básico).

También existen los llamados *dispositivos de localización*, estos generalmente son elementos que transforman coordenadas analógicas X-Y en sus correspondientes digitales, el uso más común que se da a estos dispositivos es para manejar la posición en que se efectuan las operaciones de edición; entre estos elementos los más conocidos son los *ratones*, *trackballs*, *palancas de juego* (joysticks), *tableros digitalizadores* y *pantallas sensibles al tacto* (touch screen), de estos los más populares son los ratones pues combinan también el dispositivo de elección al tener dos

o tres botones que se usan para indicar alguna operación una vez que se ha localizado la sección que se desea trabajar. Es conveniente mencionar que la mayoría de los teclados modernos han sido dotados de algunas teclas que permiten simular dispositivos de localización y de elección (las teclas para indicar movimiento en la pantalla y las teclas de funciones especiales).

Por otra parte se ha intentado usar como dispositivo de entrada algunos elementos *reconocedores de sonido*, sin embargo esto sólo ha sido a nivel experimental y aún no se tiene mucho exito.

En lo que toca a los *dispositivos de salida* (que tienen como función la representación visual de los elementos editables y el resultado de las operaciones efectuadas) los primeros fueron los llamados *teletipos*, a estos siguieron las *terminales de salida en papel* y los primeros *monitores o pantallas de video*, que eran funcionalmente iguales a las terminales de impresión pues mostraban lo mismo que en ellas. Actualmente los monitores permiten mover facilmente el cursor, insertar o borrar caracteres o líneas y mover el texto hacia arriba o hacia abajo. A últimas fechas se han desarrollado las llamadas estaciones de trabajo, que se basan en computadoras dedicadas a trabajos generalmente de diseño (en ingeniería), estas estaciones tienen monitores de muy alta resolución (high resolution raster display) y permiten manejar diferentes tipos de caracteres, con lo que se puede tener en pantalla representaciones muy realistas de lo que será el documento final.

El *lenguaje de interacción* es el que especifica la forma y el significado de las instrucciones que pueden especificarse al editor, para esto se divide en tres componentes: semántica, sintáctica y léxica.

La *componente semántica* especifica la funcionalidad del lenguaje, es decir que operaciones son válidas para cada elemento, que información necesita el sistema para llevarlas a cabo, que resultados producirá cada una y que errores pueden ocurrir.

La *componente sintáctica* especifica las reglas bajo las que los elementos atómicos se unen para formar oraciones. La sintaxis empleada debe ser simple de aprender y usar, para ésto es conveniente que sea consistente para todas las operaciones. La sintaxis de las operaciones que se pueden efectuar en el sistema de edición cae en alguna de estas tres categorías: prefija, infija y postfija. En una *sintaxis prefija* lo primero que se especifica es la instrucción deseada y después los elementos con que se trabajará. Si se usa *notación postfija* primero se especifican los elementos sobre los que se trabaja y posteriormente la operación. Finalmente en la *notación*

infija la operación a efectuar va entre los elementos sobre los que se trabaja.

En algunos lenguajes la sintaxis utilizada para especificar las instrucciones es de un solo tipo, lo que hace una interfaz más consistente; en aquellos sistemas que permiten instrucciones con diferentes clases de sintaxis una de éstas es infija (generalmente para operaciones que involucran dos operandos) y se recomienda no mezclar prefija y postfija.

La *componente léxica* especifica la forma en que la información proveniente de los dispositivos de entrada se combina para formar los elementos usados en la componente sintáctica; a la información mínima con que se trabaja se le llama *lexema*. Dependiendo de la forma en que se especifiquen los lexemas las interfaces se clasifican en : interfaces orientadas a instrucciones textuales, interfaces orientadas a teclas de funciones e interfaces orientadas a menús.

En las *interfaces orientadas a instrucciones textuales* el usuario se comunica con el sistema escribiendo texto, que indica la operación deseada. Este tipo de interfaz fue de las primeras en usarse, principalmente en los editores de línea y los editores de ventana (CANDE, ED, TECO y VUE son editores que tienen este tipo de interfaz). Esta interfaz no es muy fácil de usar, pues requiere que el usuario recuerde la forma exacta de escribir las instrucciones (lo que es muy difícil si los nombres no se han elegido bien), por otra parte hace que el sistema sea lento pues se pierde mucho tiempo escribiendo cada instrucción. Algunos de estos problemas se resolvieron al definir el nombre de cada instrucción con pocas letras, pero al ganar en rapidez se perdía claridad.

Como una alternativa más eficiente surgieron las *interfaces orientadas a teclas de funciones*, en las que cada instrucción tiene asociada una tecla especial. El hecho de tener teclas especiales asociadas a cada instrucción facilita el uso del sistema, pues el usuario puede etiquetar cada tecla con el nombre de la instrucción, de tal forma que no es necesario recordar los nombres de las instrucciones. En esta interfaz surge otro problema: se necesitaban tantas teclas especiales como diferentes instrucciones tuviera el sistema o visto de otra forma el sistema sólo podía tener tantas instrucciones como teclas especiales tuvieran las terminales (si eran terminales de diferentes tipos el problema se agravaba, pues el número de teclas especiales es diferente de una a otra y además cada tecla especial genera un código que en la mayoría de los casos sólo es común a las terminales fabricadas por la misma compañía). Una forma de resolver este problema es utilizando las teclas comunes (de texto) como teclas de función, para esto se usan algunas teclas especiales (ESCAPE, CONTROL, ALTERNATE) de manera que al usarse en combinación se genera el equivalente al código de una función especial, a este mecanismo se le

llama *sobrecarga del teclado*. En este caso no se pueden etiquetar las teclas de funciones pues se ocultaría el significado original de cada tecla; por este motivo se busca que la asociación de cada función con una tecla particular permita al usuario recordar fácilmente su significado, para tales asociaciones se siguen generalmente dos criterios:

Los nombres originales de las teclas deben dar información sobre la función asociada (la tecla I asociada a la función de inserción). A esta forma de asociar teclas y funciones se le llama *nemónica*; EMACS es un buen ejemplo de un sistema con este tipo de interfaz.

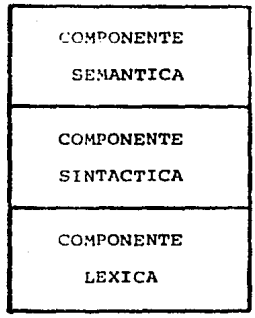
La posición de las teclas debe indicar la instrucción asociada (un ejemplo típico es la posición de las teclas que indican los movimientos del cursor en la pantalla). A esta asociación se le llama *topológica*; WORD STAR es un sistema que utiliza este tipo de interfaz.

Un aspecto que debe tenerse siempre en cuenta es que la asociación de teclas y funciones sólo debe hacerse siguiendo uno de los criterios, no hacerlo así podría resultar en una interfaz poco consistente y por lo tanto difícil de usar. De cualquier forma el uso de una interfaz orientada a funciones requiere que el usuario adquiera habilidad en el manejo de un teclado diferente al usual, lo que es difícil para algunas personas.

Como una alternativa para simplificar la interfaz del sistema con el usuario se desarrollaron *interfaces orientadas a menús*, de esta forma el usuario siempre tiene a su disposición una lista de las operaciones que puede efectuar de donde elige cúal es la deseada, en algunos sistemas las operaciones aparecen representadas por figuras que sugieren el efecto de la acción a tomar, en cuyo caso la elección se hace señalando la figura deseada. Este tipo de interfaz libera al usuario de la necesidad de memorizar la sintaxis de las diferentes instrucciones y la lista de operaciones factibles en cada nivel de operación, sin embargo tiene un gran inconveniente: el menú ocupa una zona del área visual que podría usarse para mostrar una porción mayor del documento. Como una alternativa algunos sistemas tienen interfaces orientadas a menús pero sólo cuando el usuario lo pide ("pop-up menus"), este tipo de interfaz se ha popularizado pues ya no es costoso (ni en tiempo ni en espacio) generar ventanas especiales para los menús, además para el caso de que se generen figuras se han acoplado lor ratones como los elementos que permiten seleccionar facilmente la acción deseada, ofreciendo así sistemas muy "amigables".

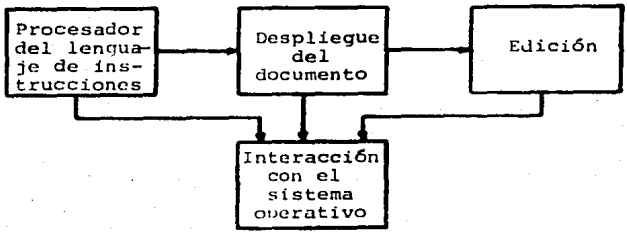
Como se ve el lenguaje de interacción se puede pensar como constituido por un conjunto de bloques, que son las diferentes componentes que se han mencionado. Esto facilita el diseño de

sistemas generales, pues no se depende de las características de los dispositivos para determinar el sistema ya que una vez definida la componente semántica (que en general es la misma para todos los sistemas de edición) se pueden mapear diferentes componentes sintácticas a ella y a su vez diferentes componentes léxicas a cada componente sintáctica.



Estructura de niveles del lenguaje de interacción

Por su parte el diseñador del sistema de edición considera que el sistema está constituido por elementos con funciones más específicas. Así a grandes rasgos se tiene un módulo *procesador del lenguaje de instrucciones*, un módulo de *despliegue del documento*, un módulo de *edición* y un módulo de *interacción con el sistema operativo*.



Componentes del sistema de edición

Las estructuras de datos básicas son dos áreas de trabajo, llamadas *"buffer" de edición* y *"buffer" de visión*. La primera se mueve a través del archivo, determinando la porción de texto que puede ser afectada por la operación actual de edición. El trabajo del editor se facilita bastante si el área de trabajo es lo suficientemente grande para almacenar todo el documento, aunque generalmente este no es el caso pues el espacio asignado al área de trabajo está limitado por las características del sistema. Para determinar sobre que porción del documento se trabaja el módulo de edición tiene un apuntador, llamado *apuntador actual de edición*. Por su parte el "buffer" de visión guarda la porción de texto que se muestra al usuario. Generalmente esta última está contenida en la primera.

El *procesador del lenguaje de instrucciones* es el módulo que efectúa la interacción con el usuario, a través de los dispositivos de entrada y usando el Lenguaje de Interacción, de manera que una vez reconocida la instrucción determina cual es la rutina semántica que debe invocarse, para ésto pasa a la componente de edición la petición del trabajo.

La *componente de edición* es la encargada de efectuar las operaciones que afectan el "buffer" de edición en su contenido, dichas operaciones se pueden clasificar como operaciones de *recorrido*, operaciones de *edición* (las que modifican el documento) y operaciones de *visualización*. Entre las operaciones básicas de edición se encuentran: *insertar* texto, *cambiar* texto, *borrar* texto, *copiar* texto, *mover* texto; estas operaciones generalmente implican operaciones de recorrido y visualización. Las operaciones de recorrido modifican la posición de los apuntadores a las áreas de trabajo y la porción de texto que se encuentra en ellas. En general estas operaciones son realizadas como parte de las operaciones de edición, sin embargo algunas veces se efectúan por petición directa del usuario; entre las posibles instrucciones de movimiento sobre el texto se tienen: *avanzar* o *retroceder* un cierto número de caracteres o líneas y la *localización* de alguna cadena especial.

El *módulo de despliegue* del documento está encargado de transformar la representación interna del documento (usada para facilitar las operaciones del editor y disminuir el tiempo empleado en transferencias del exterior) en un conjunto de símbolos comprensibles. Una de las tendencias en los nuevos editores es mejorar el diseño de este módulo, generando representaciones más cercanas a la presentación final. La relación existente entre las áreas de trabajo para edición y visión toma muchas formas siendo lo más común que en los editores de pantalla el "buffer" de visión esté contenido en el "buffer" de edición, lo que agiliza el cambio de una pantalla a otra, en este caso se tiene una señal luminosa asociada al apuntador actual de edición; a esta señal se le llama *cursor*. Originalmente el módulo de visión trataba de usar toda

la pantalla para mostrar el texto , pero actualmente las terminales tienen integradas funciones que facilitan la división de la pantalla en *ventanas*, de tal forma que en una de ellas se puede tener parte de un archivo y en otra una porción diferente (ya sea del mismo documento o de otro).

Todas la operaciones descritas anteriormente suponen que el texto se encuentra accesible, esto se logra generalmente usando las facilidades que ofrece el sistema operativo para el manejo de archivos. De estas operaciones y otras relacionadas (interacción con la terminal, etcétera) se encarga el *módulo de interacción con el sistema operativo.*

La descripción en términos de estos elementos ha sido para facilitar su comprensión, sin embargo tal división no es tan estricta pues la relación existente entre estos elementos es muy compleja, de hecho muchas de las operaciones de edición llevan implícitas funciones de todos los módulos, siendo lo más común la alteración de los diferentes "buffers", lo que a su vez implica algunas operaciones del sistema operativo.

Un aspecto que no debe dejar de considerarse por parte del diseñador de un sistema de edición es el ambiente en que funcionará. Las facilidades y restricciones que imponen los diversos tipos de sistemas de cómputo determinan algunas de las características de un sistema; por ejemplo, en un sistema operativo de *tiempo compartido* no se debe construir un programa que acapare los recursos, imposibilitando el trabajo de otros usuarios. En este sentido algunos editores aprovechan las ventajas que ofrecen las terminales inteligentes para dejar que ellas hagan parte del trabajo de edición, esta forma de trabajar también tiene sus problemas, en caso de que el sistema operativo falle no se puede asegurar que el documento refleje el estado en que el usuario lo dejó.

REFERENCIAS

La información más completa respecto a los sistemas de edición se puede encontrar en los artículos de Meyrowitz y VanDam [MEY-82a], [MEY-82b]. Otra fuente, en donde se encuentra bastante información sobre algunos editores particulares (Z, EMACS, ETUDE) son la memorias de la Conferencia sobre Manipulación de Texto [ACM-81].

Entre los artículos clásicos es conveniente revisar el de Irons [IRO-72], pues muchos de los conceptos se mencionan en este artículo por vez primera. También es conveniente revisar el libro de Estructuras de Datos de Wirth [WIR-75], en él se describe un editor de tarjetas. Además los manuales de los diferentes editores accesibles (CANDE, TECO, EDI, PERFECT WRITER, TURBO PASCAL) dan buena idea de las características de cada sistema y del tipo de editor que representan.

Los editores guiados por la sintaxis están adquiriendo bastante importancia, para un estudio más profundo sobre este tema se puede consultar [BAR-84], en este libro se trata lo relativo a sistemas interáctivos y hace bastante enfasis en estos sistemas. También se pueden consultar las memorias del simposium sobre Ingeniería de Software en los ambientes de desarrollo de Programas [ACM-84].

SISTEMA DESARROLLADO

Las características de un sistema de edición determinan en buena medida el uso que se haga de él, es por esto que al diseñar un sistema interactivo se debe tomar muy en cuenta la relación que se va a tener con el usuario (es a lo que se conoce como *Factores Humanos*), en general se trata de explotar sus capacidades proporcionándole mejores condiciones de trabajo; en ese sentido se deben considerar los siguientes elementos:

RETROALIMENTACION. El sistema debe mostrar los efectos que causó la acción indicada por el usuario. La respuesta debe ser inmediata y obvia, además de mostrarse en el lugar que él lo espere.

CONSISTENCIA. El sistema no debe ser ambiguo. Debe ser posible describir las acciones del usuario en base a un conjunto de reglas, de tal forma que una vez aprendidas en un contexto se puedan aplicar a otros. En este sentido es conveniente que dichas reglas sean tomadas de las de otros sistemas con que interactua el usuario.

MEMORIA. El sistema no debe exigir que el usuario tenga muy buena memoria, es preferible confiar en la memoria de la computadora. En este sentido son más faciles de manejar aquellos sistemas en que se especifican las instrucciones en base a menús.

*SIMPLICIDAD.*El usuario debe considerar al sistema como un elemento muy fácil de manejar, en base a operaciones muy simples.

ORIENTACION. Se debe indicar al usuario en dónde está y como regresar a donde estaba o cómo hacer alguna operación. En base a estas consideraciones es que se han hecho muy populares aquellos sistemas interactivos con *ayuda en línea.*

USUARIO. Finalmente debe considerarse qué tipo de persona es la que hará uso del sistema, esto determina el balance entre las operaciones que efectuará el usuario y las que serán responsabilidad del sistema.

En base a estas consideraciones se pueden mencionar las siguientes como características deseables en un sistema de edición:

EDITOR DE PANTALLA. La tendencia actual es a usar exclusivamente terminales de

video en las sesiones de edición. La pantalla de la terminal será una ventana al documento, lo que permitirá tener un panorama más amplio de su estado actual. Esta ventana se puede desplazar a través del archivo en dirección vertical u horizontal. Generalmente se tiene un indicador de la posición actual, éste se maneja dependiendo de las capacidades del editor (movimientos por caracter, por palabra, por línea, etcétera). En esta pantalla se debe reservar espacio para información al usuario (nombre del archivo, línea actual, etcétera), este espacio no debe ser muy grande pues es más importante mostrar una buena porción del documento.

MANEJO TRANSPARENTE DE LOS ARCHIVOS. Debe manejar los archivos de manera elegante. Si se tiene un archivo de tamaño mayor al disponible en el área de trabajo una parte estará en dicha área y la otra en memoria secundaria. Este manejo deberá ser invisible para el usuario; para él debe ser posible editar cualquier parte del archivo y nunca estará en una posición en que sea imposible pasar a otro punto del archivo.

ARCHIVO DE TRABAJO. Se debe trabajar sobre una copia del archivo a editar, de tal manera que sea posible regresar el documento a la forma en que estaba antes de editarlo. También es conveniente tener la posiblidad de generar copias con cambios a diferentes archivos sin salir de la edición.

AREAS DE TRABAJO Y VENTANAS MULTIPLES. Debe permitir la edición de más de un archivo a la vez, por lo que tendrá varias áreas de trabajo, una para cada archivo. Además debe ser capaz de mostrar estos al mismo tiempo en la pantalla (dividiéndola) y de transferir texto de un área a otra.

VELOCIDAD. La respuesta a las intrucciones que dé el usuario debe ser instantánea. También es deseable que la pantalla muestre el resultado final de una cadena de instrucciones y no los resultados parciales correspondientes a cada una de ellas; ésto es para evitar confundir a los usuarios y tener una respuesta más rápida.

DEFINICION DEL TECLADO. No es conveniente usar las teclas de funciones pues esto complica la adaptación de los mecanógrafos. Por otra parte los códigos generados por estas teclas no son comunes entre los diferentes tipos de terminal. Así, es conveniente limitarse al teclado alfanumérico y de puntuación, cuyo código es estándar. En este sentido la única posibilidad que queda para indicar las instrucciones es usando *caracteres de control;* esto es debido a la forma en que se interpretan los caracteres comunes.

INSTRUCCIONES NEMOTECNICAS. Las instrucciones nemotécnicas son más fáciles de recordar y tienden a distribuirse mejor sobre el teclado; de esta forma es más rápida y fácil la comunicación del usuario con el sistema, pues puede usar ambas manos para especificar las instrucciones.

INSERCION O REEMPLAZO AUTOMATICO. Desde el momento de iniciar la sesión de edición se debe permitir alguno de estos modos de trabajo. Si se tiene inserción automática todo el texto que se teclee (caracteres alfanuméricos y de puntuación) se inserta en la posición que marca el cursor, recorriendo el texto original y el cursor tantos lugares como caracteres se hayan insertado. En caso de tener reemplazo automático cada caracter tecleado sustituye al caracter bajo el cursor, avanzando el cursor una posición. En la mayoría de los editores de pantalla el modo de inserción automática es el usado desde un principio. Sin importar cual sea el modo inicial de trabajo debe ser posible trabajar de la otra forma.

RECUPERACION DE LO BORRADO. Se debe proporcionar algún mecanismo para que el usuario recupere el texto que se haya borrado indebidamente durante su sesión de edición. En algunos sistemas este mecanismo se usa también para copiar o transferir un bloque de texto.

SALVAR EL ESTADO DE EDICION. Es conveniente tener la posibilidad de suspender la edición en cualquier momento para hacer algún otro trabajo con el sistema operativo y continuar después como si no hubiera ocurrido nada.

ADAPTABILIDAD. El usuario debe tener la posibilidad de adaptar el sistema de edición para facilitar su trabajo específico. Esto incluye la posibilidad de definir el tipo de objetos con que trabajaran las operaciones del editor; por ejemplo si se trabaja con programas se pueden definir procedimientos, proposiciones y expresiones como objetos de edición. También debe ser posible cambiar la configuración de las instrucciones pudiendo definir nuevos nombres para ellas, al gusto del usuario.

MANEJO DE ERRORES. No se debe ignorar que todo usuario puede cometer errores, por lo que el editor debe estar preparado para actuar de acuerdo al tipo de error cometido; por ejemplo, en caso de una instrucción ilegal (por no existir o no ser válida en cierto contexto) es conveniente usar la bocina para llamar la atención del usuario y mostrar un mensaje en el área de información (por cierto los mensajes del sistema al usuario no deben

incomodarlo jamás, por lo que deben ser claros, concisos y diplomáticos).

El sistema desarrollado no tiene todas las características del editor ideal, en general dotarlo de tales características implica crear un programa muy grande, lo que no es conveniente para un sistema en un ambiente de tiempo compartido en una computadora con poca memoria. Un aspecto que se consideró fue la posibilidad de aumentar las operaciones de edición que se pudieran efectuar, de manera que en aspectos de edición el sistema fuera robusto.

Algunas de estas características no son fáciles de realizar, para hacerlo se tuvieron que definir estructuras de datos especiales y los algoritmos adecuados a ellas. Para simplificar la descripción del sistema se describen dichas estructuras y algoritmos en terminos de cada una de las características del sistema.

MANEJO DE LA PANTALLA. El manejo de la pantalla es responsabilidad del módulo de despliegue. Para que la pantalla refleje la versión actual del estado del texto es conveniente que la terminal haga parte del trabajo. En algunos sistemas de edición (sobre todo en computadoras personales) el manejo de la pantalla se hace sobre una zona especial de la memoria, esta zona es considerada el "buffer" de visión, de manera que para reflejar el resultado de alguna operación basta con copiar la parte afectada del "buffer" de edición al de visión; en este tipo de sistemas el manejo de la pantalla pasa inadvertido para el usuario. En el sistema desarrollado el área de trabajo del módulo de visión está contenida en la del módulo de edición pero no se tienen tales facilidades y las operaciones para la actualización de la visión se basan en las funciones que poseen las terminales; para esto toda terminal debe poseer las siguientes características:

Posibilidad de mover el cursor en cualquier dirección a partir de la posición en que se encuentre (movimientos relativos).

Posibilidad de colocar el cursor en cualquier parte de la pantalla (movimientos absolutos).

Facilidad de limpiar la pantalla, ya sea en su totalidad o de la posición en que se encuentre el cursor hasta el final de la pantalla o de la línea.

Posibilidad de insertar una línea en blanco en la posición que marque el cursor. Como efecto de esta operación las líneas comprendidas entre el cursor y el final de la pantalla se desplazan un lugar hacia abajo, perdiéndose la última línea.

Posibilidad de borrar la línea en que se encuentre el cursor. En este caso las líneas que se encuentren por debajo del cursor se desplazan un lugar hacia arriba, dejando una línea en blanco al final de la pantalla (es responsabilidad del sistema llenar esta última línea con el texto adecuado).

Posibilidad de insertar un caracter en la posición que marque el cursor, los caracteres que estén en la misma línea y hacia la derecha a partir del cursor se mueven un lugar a la derecha, perdiéndose el último de ellos.

Posibilidad de borrar el caracter señalado por el cursor, en este caso los caracteres que se encuentran a la derecha del cursor se recorren un lugar a la izquierda, quedando un espacio en blanco en la última posición (tambien es responsabilidad del sistema llenar este lugar con la información adecuada).

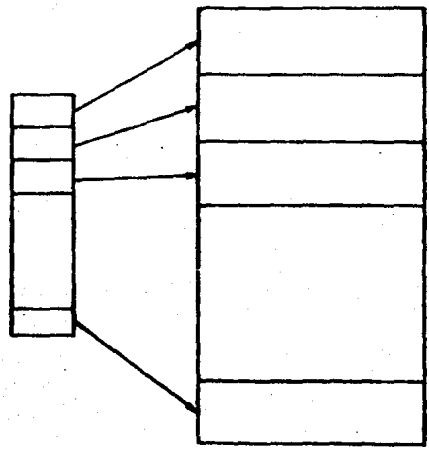
De las características anteriores las de inserción o borrado de líneas o caracteres son las más importantes, pues en caso de no disponer de ellas la operación de insertar un caracter implica la restauración de la pantalla desde la posición del cursor hasta el final de la línea, lo que a su vez implica enviar a la terminal cadenas de caracteres más grandes. Estos problemas se presentan también al insertar líneas o borrar; para el usuario el sistema tarda en responder a sus operaciones.

Como los códigos con que se especifican las funciones que reconoce la terminal son diferentes de un tipo de terminal a otra se tienen definidas dos tablas, para poder manejar dos tipos diferentes de terminales; al iniciar la sesión se pregunta al usuario cúal es el tipo de terminal que desea usar. Como se verá más adelante esta definición de la terminal también afecta la definición del teclado.

MANEJO DE ARCHIVOS. El área de trabajo del módulo de edición tiene un tamaño de 4800 caracteres (60 líneas de 80 caracteres), de manera que no todos los archivos pueden tener en ella todo el texto. Para poder hacer uso más eficiente de los archivos (menos operaciones de entrada o salida) es preferible hacer accesos directos al archivo, pero en el sistema operativo en que se trabajó se necesita que los archivos tengan registros de longitud fija; como el tipo de archivo estándar tiene registros de longitud variable se elaboró un módulo que copia el archivo original en otro con registros de longitud fija (originalmente los registros son de 80 caracteres). Al generar la copia del archivo se mantiene intacto el documento original, la única forma de borrarlo es bajo petición explícita del usuario. Por otra parte el archivo de salida debe tener la

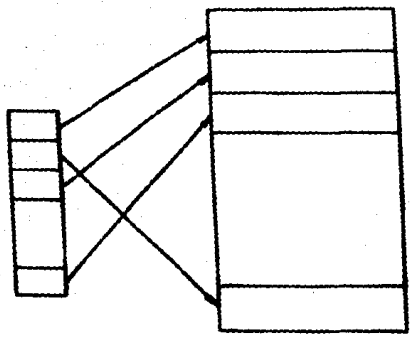
misma estructura del archivo original por lo que existe otro módulo que transforma un archivo con registros de longitud fija en otro con registros de longitud variable.

El módulo responsable de tener en el área de trabajo la parte necesaria es el módulo de recorrido. Para poder llevar a cabo este trabajo parte el archivo en conjuntos de líneas contiguas, originalmente sólo hay dos bloques: el que corresponde a la sección cargada en el área de trabajo y la parte restante; conforme se avanza en el proceso de edición el archivo se va partiendo en más bloques. Para llevar un registro de estos bloques se tiene un arreglo en donde se determina cada bloque.



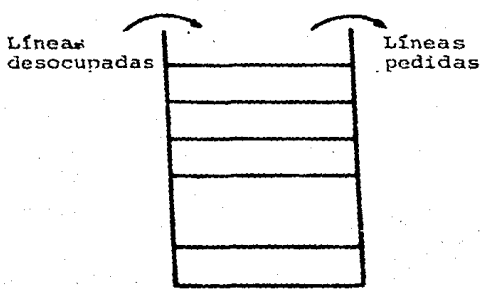
División del archivo en bloques

Esta estructura permite manejar el archivo con menos operaciones de transferencia, pues en caso de borrar o insertar líneas no es necesario reestructurar todo el archivo, sólo se hace en el bloque correspondiente y en caso de no ser suficiente el espacio ocupado originalmente por dicho bloque éste se añade al final del archivo. De esta forma los bloques están desordenados físicamente, pero lógicamente están en el orden deseado.



Archivo con bloques desordenados

AREAS DE TRABAJO Y VENTANAS MULTIPLES. El área de trabajo está diseñada para que pueda manejar más de un archivo a la vez, también permite insertar o borrar líneas sin demasiados movimientos. Para lograr esto se tiene un administrador del área de trabajo, éste divide la memoria asignada en un conjunto de líneas y las coloca en una *lista de espacio disponible*, de esta forma cada vez que se necesita una nueva línea se toma de dicha lista. El administrador de la memoria también se encarga de que las líneas que son borradas del texto pasen a la lista de espacio disponible. Esta lista se maneja como una pila, la última línea insertada es la primera en salir.



Lista de espacio disponible

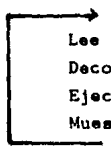
El área de trabajo del módulo de edición es también una lista, de manera que lógicamente se tiene un conjunto de líneas contiguas pero físicamente están separadas. El hecho de que sea una lista permite que en una misma área de memoria física se encuentre más de un archivo a la vez, pues para cada uno sólo basta tener la cabeza de la lista. En la versión actual sólo se maneja un archivo pero no es demasiado problema extenderlo a manejar dos o más.

El área de trabajo tiene más líneas de las que caben en una pantalla, esto es para agilizar el cambio de la porción de texto que se muestra en la pantalla a la sección anterior o posterior. Por otra parte en caso de que el usuario agote las líneas disponibles o se acerque a alguno de los extremos del bloque usado el sistema invoca al módulo de recorrido para que actualice el área de trabajo con otro bloque del archivo.

Como ya se mencionó el manejo de la pantalla es responsabilidad del módulo de visión, en caso de que se esté trabajando con varios archivos está contemplada la posibilidad de mostrar al menos dos de ellos simultáneamente. Para esto se parte la pantalla con una línea horizontal, dejando en la mitad inferior el archivo con que se esté trabajando en ese momento. Las operaciones que se pueden hacer entre estas zonas de trabajo están restringidas a transferencias de porciones del texto mostrado en una de ellas a la otra. Futuras versiones del sistema de edición pueden eliminar este tipo de restricciones.

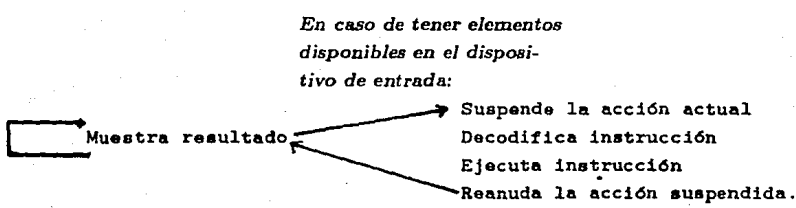
VELOCIDAD. En general el trabajo de edición se hace rápidamente, pues las operaciones en sí no son muy complejas y los algoritmos con que se realizan tampoco. Sin embargo la respuesta del sistema se siente lenta. Para entender porqué el sistema no es rápido en su respuesta es conveniente revisar brevemente su funcionamiento.

El sistema se encuentra siempre en una iteración:



Como se ve para que la pantalla muestre el resultado de alguna operación es necesario que dicha instrucción haya pasado por las tres étapas anteriores. De éstas la de lectura es la que tarda más tiempo en efectuarse, pues depende de la velocidad con que el usuario escriba la instrucción y sea transferida de la terminal a la computadora. La forma más eficiente de

programar este tipo de sistemas es apegandose al siguiente esquema:



Obviamente la computadora no debe estar preguntando si es necesario suspender el proceso actual (mostrar el resultado), lo mejor es manejar estas situaciones en base a un mecanismo de *manejo de interrupciones;* de esta forma el sistema siempre está trabajando y en el momento en que ocurra el evento (hay datos en la terminal) se suspende lo que se estaba haciendo y se atiende la interrupción (toma el dato de entrada y ve si ya hay alguna instrucción por procesar), una vez atendida la interrupción continua con el proceso suspendido.

Al usar este mecanismo la respuesta al usuario es instantánea, lamentablemente no siempre es posible llevarla a la práctica. En primer lugar es necesario tener instrucciones que permitan leer datos en caso de que los haya, lo que en general no se tiene en los lenguajes de programación de alto nivel (PASCAL, C); la mayoría de las veces esta parte se tiene que programar a nivel de *lenguaje ensamblador.* Estas instrucciones están limitadas por las características del sistema operativo, mientras más grande y complejo sea son menos las facilidades que tiene el usuario de alterar su configuración. Por otra parte el hecho de manejar código reentrante es otra limitante, pues el código de las operaciones de entrada y salida también debe ser reentrante. Bajo estas condiciones el manejo de las interrupciones puede ocasionar algunos problemas, por lo que se decidió apegarse al primero de los esquemas. El problema del manejo de interrupciones usando código reentrante puede ser tema de otros trabajos.

En lo que respecta al manejo de cadenas de instrucciones el sistema las ejecuta por separado, el algoritmo que se usa no revisa si aún hay trabajo por hacer antes de mostrar un resultado, para lograrlo es necesario trabajar con el esquema de manejo de interrupciones; de esta forma se puede crear una cola de instrucciones por ejecutar y tratar de atenderlas antes de modificar la pantalla.

TECLADO. Para especificar las instrucciones se utilizan las teclas de control, el usar estas

teclas limita el número de instrucciones diferentes al número de teclas de control que se tienen (32); de estas teclas algunas (aproximadamente ocho) están reservadas por tener un significado especial ([RETURN] ≈ [CTRL] - [M] ; [LF] ≈ [CTRL] - [J] ; [TAB] ≈ [CTRL] - [I]) lo que limita aún más el número de caracteres disponibles. En la versión actual no se tienen muchas instrucciones, pero en caso de ser necesario se puede usar uno de los caracteres de control (ESCAPE) para indicar instrucciones formadas por cadenas de más de un caracter de control. El hecho de usar de esta forma el teclado asegura que el sistema funciona igual en cualquier tipo de terminal, sin embargo para el usuario común es difícil aceptar tener teclas especiales en su terminal y no poder usarlas. Ya se ha mencionado el problema que representan las teclas especiales pero considerando que sería muy poco confortable un sistema de edición que no permitiera usar las teclas de manipulación del cursor se decidió aceptarlas para indicar el movimiento del cursor. Esto limita todavía más las teclas de control disponibles, además de que en algunas terminales las teclas de movimientos del cursor no generan un caracter de control, sino una cadena (2 ó 3), lo que complica el manejo de los caracteres de entrada y el reconocimiento de las instrucciones indicadas por el usuario.

INSTRUCCIONES NEMOTECNICAS. Se decidió usar una asociación nemotécnica para las teclas que indican las operaciones de edición. Como ya se mencionó los caracteres de control disponibles son pocos y por otra parte los nombres de varias operaciones empiezan con la misma letra, de manera que la asignación de caracteres de control a cada operación fué un poco difícil. Las operaciones que reconoce el sistema y los caracteres de control asociados a cada una son:

CARACTER DE CONTROL	INSTRUCCION
[CTRL] - [A]	Avanza al siguiente bloque de 24 líneas (una pantalla).
[CTRL] - [B]	Baja el cursor una línea, si en esta línea el último caracter está a la izquierda de la posición actual el cursor se ajusta a dicha columna.
[CTRL] - [D]	Define el tipo de terminal que se está usando.
[CTRL] - [E]	Elimina la línea en que se encuentra el cursor.
[CTRL] - [F]	Finaliza la sesión de edición y genera un archivo con el texto editado.
[CTRL] - [L]	Localiza una cadena de texto.
[CTRL] - [N] o [RETURN]	Inserta una línea en blanco, el cursor se posiciona bajo el primer caracter distinto de blanco (indentación automática).
[CTRL] - [N]	Define el Nombre del archivo a editar. En caso de que dicho archivo no exista lo crea con el nombre especificado.

CARACTER DE CONTROL	INSTRUCCION
CTRL - R	Restaura la pantalla.
CTRL - B	Sube el cursor una línea.
CTRL - V	Muestra la Ventana que contiene las 24 líneas anteriores.
DEL	Borra el caracter anterior al cursor.
→	Avanza el cursor una posición.
←	Retrocede el cursor una posición.
↑	Sube el cursor una línea.
↓	Baja el cursor una línea.

INSERCION AUTOMATICA. Mientras no se haya especificado alguna instrucción todo aquel caracter que no sea de control (alfanuméricos y de puntuación) es interpretado como un caracter que se debe insertar en la posición que marque el cursor. Actualmente no se proporciona la posibilidad de reemplazar automáticamente el caracter que se encuentre bajo el cursor, pero no es difícil añadir esta característica al sistema.

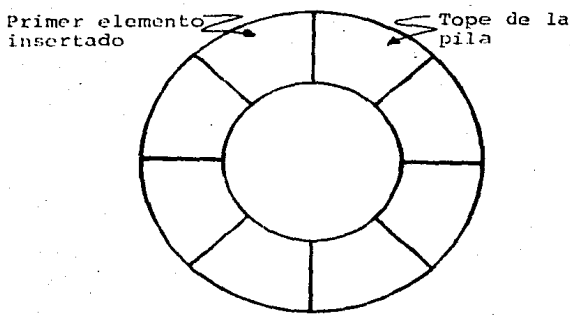
MANEJO DE ERRORES. Actualmente el sistema sólo reconoce como errores los caracteres de control que no están asociados a ninguna instrucción. Otro tipo de errores no están considerados, pues la mayoría de los efectos causados por las operaciones de edición se pueden revertir. Por ejemplo:

Si se insertó texto de más se puede borrar.

Si se movió el cursor a algún lugar no deseado se puede regresar.

En realidad de las operaciones de edición la de borrar texto es la más difícil de anular, pues no es fácil recordar exactamente todo el documento, por otra parte aun recordando el texto borrado es mucho el tiempo que se desperdicia al volverlo a insertar; como se ve es esta operación la que debe ser posible revertir. Una forma de manejar esta operación para poder recuperar lo borrado es estableciendo una *zona de texto borrado*, en esta zona se coloca el texto conforme se vaya borrando. La mejor forma de manejar el texto borrado es usando una *pila circular*, de manera que los últimos elementos borrados sean recuperables. En realidad es preferible manejar dos estructuras, una para caracteres y la otra para líneas. El hecho de manejarlas

como estructuras circulares permite limitar las posibilidades de recuperación a los elementos que se hayan borrado más recientemente.



Pila circular

Un aspecto que es importante mencionar es que el mecanismo más sofisticado para el manejo de errores (en este caso recuperación del texto borrado) no funciona para todos los casos, siempre hay alguna forma de trabajar de manera que el sistema no pueda ayudar a enmendar el error.

SALVAR EL ESTADO DE EDICION. Para poder salvar el estado de edición es necesario tener una historia del trabajo realizado, de manera que al reanudar la sesión se tome de ahí la información que coloque al sistema en el punto en que se le dejó.

Uno de los aspectos que influyó en la decisión de no permitir estas operaciones fué el tamaño del área de datos que se ocuparía, pues por cada activación del editor se carga en memoria el bloque correspondiente al área de datos del usuario. Como se ha mencionado, en este sistema lo que se comparte es el área de código no la zona de datos de manera que crear un programa con un área de datos grande va contra la filosofía de diseño de este sistema.

ADAPTABILIDAD. Uno de los aspectos que influyen notablemente en el uso de un sistema de edición es la comodidad que se proporcione al usuario; son más usados aquellos editores en que el trabajo del usuario es muy simple. Por ejemplo al editar un programa es muy comodo tener la posibilidad de encontrar con una sola instrucción la palabra END correspondiente a una

palabra BEGIN. Para lograr este tipo de operaciones hay dos posibilidades: configurar el sistema para un tipo de aplicaciones o permitir que el usuario lo configure a su gusto. Obviamente la segunda de estas alternativas es más atractiva para el usuario, sin embargo los problemas para llevarla a la práctica son muchos. Esta característica no está incluida en la primer versión del sistema pero conforme se vaya definiendo el tipo de aplicaciones en que se usa el sistema se podrá determinar la forma que debe tomar con respecto a su adaptabilidad, pues las estructuras necesarias dependen del tipo de objetos que se manejen.

La mayoría de los algoritmos empleados son simples y fáciles de programar en un *lenguaje de programación de alto nivel* (ALGOL, PASCAL, C), pero las características del sistema (debía ser un editor con código compartido) restringían el conjunto de posibles lenguajes; de hecho para asegurar que se hacen los direccionamientos en la forma que lo exige el código reentrante parece ser inevitable programar en el *lenguaje ensamblador* de la computadora (MACRO-11). Afortunadamente el compilador del lenguaje C no genera código de máquina directamente, como lo hace el compilador de PASCAL, genera un programa en lenguaje ensamblador; de manera que se eligió el lenguaje de programación C para desarrollar el sistema.

De todo el sistema la mayor parte (aproximadamente 80 %) está escrito en C y el resto en lenguaje ensamblador; esta última parte corresponde principalmente a las rutinas de entrada y salida. El haber trabajado en un lenguaje de alto nivel permitió desarrollar el sistema en un tiempo relativamente corto, pues además de ser más pequeñas las rutinas son también más legibles y por lo tanto menos suceptibles a errores.

Cada una de las rutinas que se programó en C tuvo que pasar por un proceso posterior a la compilación, pues el programa generado en lenguaje ensamblador debía ser editado para cambiar los modos de direccionamiento principalmente. La mayoría de las rutinas fueron modificadas fácilmente, pues las estructuras de control del lenguaje C se traducen a estructuras muy simples en lenguaje ensamblador. La única dificultad se presentó al traducir las estructuras CASE, pues el compilador construye una tabla de direcciones relativas, pero una vez determinada la manera de traducir las direcciones relativas a direcciones absolutas no hubo mayor problema. En el anexo A se muestra una rutina en el lenguaje de programación C, el equivalente en lenguaje ensamblador generado por el compilador y la forma en que quedó después de haber sido modificada para poder ser compartida.

REFERENCIAS

La descripción del editor ideal está basada en los conceptos mencionados en los artículos de Meyrowitz y VanDam [MEY-82a], [MEY-82b]. Algunos elementos, sobre todo lo relativo a los Factores Humanos, están tomados de diversos artículos [FIN-82], [JON-82] y [SIM-82].

Respecto a los algoritmos de edición las ideas generales están en el libro de Kernighan y Plauger [KER-78]; algunas de las estructuras y algoritmos asociados son ideas tomadas de otro tipo de aplicaciones (por ejemplo el administrador del área de edición se basa en la idea de administrador de memoria usada en sistemas operativos).

Es interesante revisar la forma en que algunas componentes de un sistema de edición se han desarrollado en sistemas muy conocidos (EMACS, ETUDE, Z), la información disponible se encuentra en las memorias de la primera conferencia sobre manipulación de texto [ACM-81].

CONCLUSIONES

Se ha mostrado un procedimiento efectivo para construir programas con código compartido en un sistema operativo que originalmente no contempla este tipo de programas. Este procedimiento, que permite usar la memoria principal de manera más eficiente, es aún más efectivo que el método propuesto por los diseñadores del sistema, pues el código sólo permanece en memoria cuando lo necesite; a diferencia de las bibliotecas residentes, que siempre ocupan la memoria, aun si estas bibliotecas no se usan.

Para mostrar el impacto que tienen los programas compartidos se elaboró un editor de textos con código compartido, pues mucho del trabajo que se realiza en el sistema de cómputo del Departamento de Matemáticas de la Facultad de Ciencias es de edición de textos. Además de reducir el número de transferencias a memoria externa este nuevo sistema de edición posee características que hacen más agradable y simple el trabajo de edición, pues fué diseñado considerando las características del editor ideal.

En el anexo B se muestran dos mapas de la memoria, el primero cuando cinco usuarios trabajan con el sistema de edición usado antes de este trabajo y el segundo cuando el mismo número de usuarios trabajan con el nuevo sistema. Como se ve el espacio de memoria utilizado es mucho menor, lo que deja una buena cantidad de memoria para otro tipo de trabajos.

Por otra parte este trabajo sienta las bases para otro tipo de trabajos. Por ejemplo se puede experimentar en sistemas operativos reales. En este sentido puede pensarse en una modificación al sistema operativo para que la generación de código compartible sea su responsabilidad y no del usuario (para esto deben modificarse el constructor de tareas y el cargador). También puede pensarse en incorporar un mejor manejo de las operaciones de entrada, de tal forma que se tenga un mecanismo de manejo de interrupciones con código compartible (esto redundaría en un sistema con mejor respuesta al usuario). El sistema de edición también puede ser extendido, primordialmente en las operaciones de edición que puede efectuar.

ANEXO A

```
>
>TYPE -P E1.C
#include <stdio.h>
#include "def.c"

edita(accion,obj)
int    accion ;
char   obj ;
{
   register  int  stat ;
   switch ( accion )
   {
     case INS :
                stat = inserta(obj) ;
                break ;
     case NULIN:
                stat = inslin() ;
                break ;
     case INSTAB:
                stat = metetab() ;
                break ;

     default:
                stat = ERROR ;
                break ;
   }
   return(stat) ;
}
E1.C listado
>
```

```
>TYPE -P E1.S
/ Decus C Patch level 9, Compilation date: 'Tue Sep 10 18:21:01 1985'
        .psect  c~data
        .psect  c~code
_edita:
        jsr     r5,csv~
        mov     4(r5),r0
        br      .3
.4:
        mov     6(r5),-(sp)
        jsr     pc,_inserta
        tst     (sp)+
        mov     r0,r4
        br      .2
.5:
        jsr     pc,_inslin
        mov     r0,r4
        br      .2
.6:
        jsr     pc,_metetab
        mov     r0,r4
        br      .2
.7:
        clr     r4
        br      .2
.3:
        dec     r0
        cmp     r0,$2
        bhi     .7
        asl     r0
        jmp     *.77777(r0)
        .psect  c~mwcn
.77777:
        .word   .4,.5,.6
        .psect  c~code
.2:
        mov     r4,r0
.1:
        jmp     cret~
        .psect  c~data
        .even
        .globl  _edita
        .globl  _inslin
        .globl  _metetab
        .globl  _inserta
        .globl  csv~
        .globl  cret~
        .psect  c~code
        .even
        .psect  c~data
        .even
        .psect  c~mwcn
        .even
E1.S listado
>
```

```
>TYPE -P E1.MAC
        .title  edita
!/ Decus C patch level 9, Compilation date: "Tue Sep 10 18:21:01 1985"
        .psect  $$resk
edita:
        jsr     r5,csv$
        mov     4(r5),r0
        br      .3
.4:
        mov     6(r5),-(sp)
        jsr     pc,inserta
        tst     (sp)+
        mov     r0,r4
        br      .2
.5:
        jsr     pc,inslin
        mov     r0,r4
        br      .2
.6:
        jsr     pc,metetab
        mov     r0,r4
        br      .2
.7:
        clr     r4
        br      .2
.3:
        mov     pc,     r1
        add     #.77777-.,r1
        dec     r0
        cmp     r0,#10
        bhi     .14
        asl     r0
        add     r1,     r0
        mov     (r0),   r1
        add     pc,     r1
        jmp     (r1)
.77777:
x       =       .-2
        .word   .4-x,.5-x,.6-x

.2:
        mov     r4,r0
.1:
        jmp     cret$
        .even
        .globl  edita
        .globl  inslin
        .globl  metetab
        .globl  inserta
        .globl  csv$
        .globl  cret$
        .psect  $$resk
        .even
        .even
        .end
E1.MAC listado
>
```

ANEXO B

```
>PAR
LDR     000000 000000 MAIN TASK
SYSPAR  070000 010000 MAIN TASK
FCPPAR  100000 026000 MAIN TASK
GEN     126000 632000 MAIN SYS
        126000 061400 SUB  (...VUE)
        207400 061400 SUB  (VUET5 )
        271000 061400 SUB  (VUET3 )
        352400 061400 SUB  (VUET6 )
        434000 061400 SUB  (VUET7 )
        515400 010000 SUB  (...SYS)
>
>
>
>
>PAR
LDR     000000 000000 MAIN TASK
SYSPAR  070000 010000 MAIN TASK
FCPPAR  100000 026000 MAIN TASK
GEN     126000 632000 MAIN SYS
        126000 027000 SUB  (...EDS)
REGCOM  155000 025700 SUB  DYNAMIC
        202700 010000 SUB  (...SYS)
>
>
>
>
>PAR
LDR     000000 000000 MAIN TASK
SYSPAR  070000 010000 MAIN TASK
FCPPAR  100000 026000 MAIN TASK
GEN     126000 632000 MAIN SYS
        126000 027000 SUB  (...EDS)
REGCOM  155000 025700 SUB  DYNAMIC
        202700 027000 SUB  (EDST7 )
        231700 027000 SUB  (EDST6 )
        260700 027000 SUB  (EDST4 )
        307700 027000 SUB  (EDST3 )
        336700 010000 SUB  (...SYS)
>
>
>
>PAR
LDR     000000 000000 MAIN TASK
SYSPAR  070000 010000 MAIN TASK
FCPPAR  100000 026000 MAIN TASK
GEN     126000 632000 MAIN SYS
        126000 010000 SUB  (...SYS)
REGCOM  155000 025700 SUB  DYNAMIC
        260700 027000 SUB  (EDST4 )
>
```

BIBLIOGRAFIA

[ACM-81] *Proceedings of the ACM SIGPLAN/SIGOA symposium on Text Manipulation (Portland Oregon).* SIGPLAN Notices V. 16 No. 6. Junio 1981.

[ACM-84] *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments (Pittsburg, Pennsylvania).* SIGPLAN Notices V. 19 No. 5. Mayo 1984.

[BAR-84] Barstow, David et. al. *Interactive Programming Environments.* McGraw-Hill. 1984.

[CAL-82] Calingaert, Peter. *Operating Systems Elements. A user's perspective.* Prentice-Hall. 1982.

[DAV-82] Davies, D. Julian. *String Searching in Text Editors.* Software Practice and Experience. V. 12 No. 8 pp. 709–717. Agosto 1982.

[DEC-76] Digital Equipment Corporation. *PDP-11/34 Processor Handbook.* DEC. 1976.

[DEC-77a] Digital Equipment Corporation. *RSX-11M System Reference Manual.* DEC. 1977.

[DEC-77b] Digital Equipment Corporation. *RSX-11M Executive Reference Manual.* DEC. 1977.

[DEC-77c] Digital Equipment Corporation. *RSX-11M Task Builder Reference Manual.* DEC. 1977.

[FIN-82] Finseth, Craig A. *Managing Words: What capabilities should you have with a Text Editor?* BYTE V. 7 No. 4 pp. 302–310. Abril 1982.

[GRI-78] Gries, David.
Compiler Construction for Digital Computers.
Wiley International. 1978.

[HAM-84] Hamacher, V. Carl, Vranesic, X. and Zaky
Computer Organization. 2nd Ed.
McGraw Hill. 1984.

[HSI-75] Hsiao, David K.
Systems Programming. Concepts of Operating and Data Base Systems.
Addison Wesley. 1975.

[HUN-82] Hunter, J. Alan and Hall, Nigell
A Network Screen Editor Implementation.
Software Practice and Experience.
V. 12 No. 9 pp. 843-856. Septiembre 1982.

[IRO-72] Irons, E. T. and Djorup, F. M.
A CRT Editing System.
Communications of the ACM.
V. 15 No. 1 pp. 16-20. Enero 1972.

[JON-82] Jong, Steven.
Designing a Text Editor? The user comes first.
BYTE V. 7 No. 4 pp. 284-300. Abril 1982.

[KER-76] Kernighan, Brian and Plauger, P. J.
Sotware Tools.
Addison Wesley. 1976.

[LEV-82] Levinson, Michael.
A Programmable Text Editing System.
Software Practice and Experience.
V. 12. No. 7 pp. 611-621. Julio 1982.

[LIS-81] Lister, Andrew M.
Fundamentals of Operating Systems.
MacMillan Press. 1981.

[LOR-81] Lorin, Harold and Deitel, Harvey.
Operating Systems.
Addison Wesley. 1981.

[MAC-81] MacEwen, Glenn H.
Introduction to Computer Systems.
McGraw-Hill. 1981.

[MEY-82a] Meyrowitz, Norman and VanDam, Andries.
Interactive Editing Systems: Part I.
Computing Surveys. V. 14 No. 3 pp. 50–100. Septiembre 1982.

[MEY-82b] Meyrowitz, Norman and VanDam, Andries.
Interactive Editing Systems: Part II.
Computing Surveys. V. 14 No. 3 pp. 50–100. Septiembre 1982.

[SIM-82] Simpson, Henry.
A Human-factors Style Guide for Program Design.
BYTE V. 7 No. 4 pp. 284–300. Abril 1982.

[STO-75] Stone, Harold and Siewiorek, Daniel.
Introduction to Computer Organization and Data Structures: PDP-11 Edition.
McGraw-Hill. 1975.

[WAK-81] Wakerly, John F.
Microcomputer Architecture and Programming.
John Wiley and Sons. 1981.

[WIR-75] Wirth, Nicklaus.
Algorithms + Data Structures = Programs.
Prentice-Hall. 1975.