64
29.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DISPERSIÓN DINÁMICA VIRTUAL

TESIS

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTAN:

PEDRO LÓPEZ URZÚA

ALFREDO F. OJEDA BAÑOS

DIRECTOR DE TESIS:

FIS. RAYMUNDO HUGO RANGEL GTEZ.

MÉXICO, D.F 1998

Agradecimientos

Estas palabras son un pequeño tributo para todas aquellas personas, tanto familiares como amigos, que nos ayudaron a largo de este proceso. Además, de agradecer por la paciencia otorgada, dado que no desesperaron por el tiempo empleado para la realización de este trabajo.

Tambien queremos hacer extensivo nuestro agradecimiento a las personas que ya no se encuentran entre nosotros. Pues no alcanzaron a ver la culminación de tantos años de trabajo y estudio.

Valgan tambien estas palabras para aquellas personas que no creyeron en nosotros, pues de alguna manera nos impulsaron a continuar con nuestro trabajo, de no dejarlo botado, de no desistir. Para todas ellas tambien nuestra gratitud y reconocimiento.

"Ninguna obra que lleve el sello de la razón y que haya sido emprendida para ampliar su poder puede quedar perdida del todo en el progreso de los siglos."

Fichte.

Le doy las gracias a ...

Dios por brindarme la oportunidad de estar vivo y de vivir rodeado de gente maravillosa.

mi esposa Ma. Teresa, por ayudarme a concluir una etapa más en vida, y de que juntos empecemos otra. Por brindarme lo mejor de tí a través de tus sonrisas y bromas, pues gracias a ellas, haces de mi existencia un mundo maravilloso.

mis padres, Angelina y Ladislao, por todos sus esfuerzos y sacrificios. Ustedes, mis mejores amigos, los cuales están siempre conmigo en las buenas y en las malas rachas.

mi hermana Ana Ma., que no has dejado de apoyarme, que siempre me brindas tu mejor sonrisa y haces que cualquier momento que pase a tu lado sea algo diferente, único.

la familia Demay Pérez, por todas sus enseñanzas; pues fueron un complemento ideal a mi persona. Además de toda su paciencia para la conclusión de este trabajo.

la familia Parra Velasco, por aceptarme y brindarme día con día todas sus muestras de cariño.

Alfredo, por todas tus locuras y de las cuales me haces ser partícipe de ellas.

Irma y Gerardo, por su amistad, apoyo y colaboración a lo largo de la carrera y más allá.

Victor y Grethel, por toda su ayuda desinteresada en la elaboración de este trabajo.

Agustín y Jose Luis, dos grandes camaradas que me ayudan y que soportan todas mis garrafaladas.

la gente que labora en el Depto. de Ing. de Control, por todas las facilidades otorgadas para la elaboración de este trabajo, como tambien de sus comentarios para la misma. En especial a Juan Manuel, pues es a él al que le debo toda su colaboración y ayuda.

la Universidad, por todo lo que engloba y de todo lo que me permitio utilizar para mi formación.

Pedro

"Los imposibles de hoy serán posibles mañana."

Konstantin Tsiolkovsky

Agradezco ...

A *Dios*, mi señor y creador por mi existencia, y la oportunidad de disfrutar del sol, el aire, el calor de los seres humanos y también de la frialdad del mundo.

A *Lety*, amiga, compañera, esposa cariñosa y fiel, te brindo esta tesis como fruto de la culminación de una meta muy añorada, gracias por tu amor y comprensión en los momentos de duda y desesperación, pero sobre todo gracias por ser tu la inspiradora de mis esfuerzos.

A *Anakaren*, pequeño angelito lleno de inquietud, te quiero mucho hija.

A *Ti*, que me has tenido pendiente de tu desarrollo los últimos siete meses y que aunque todavía no se como te llamaremos, ya eres parte de este esfuerzo y también de este triunfo.

A *Mis Padres, Doña Adelfa, y Don Sabas*, mis primeros formadores y amigos, quienes a pesar de todo me han dado su cariño, apoyo y han creído en mis ideas, sueños e ilusiones y a quienes debo el don de la vida; gracias por su preocupación y desvelos, que hoy se convierten en un profesionista.

A *Mis Hermanas, Gina Y Cristina*, fieles compañeras y amigas de toda mi vida, gracias por todo el apoyo, y por todo aquello que me han brindado, gracias de todo corazón por creer que el niño se podía convertir en hombre.

A *Mi Abuelita, Doña Raquel*, que aunque ya no estas físicamente aquí, se que te llena de alegría este momento, y que en donde quiera que te encuentres, quiero decirte que tu eres una gran parte de mi, en mi fortaleza y espíritu de lucha.

A *Mama Alice*, por su cariño, comprensión y consejos, gracias por tu animo y aliento cuando me lo has brindado, pero sobre todo gracias por brindarme tu corazón de madre.

A *Pedro*, el mejor amigo, compañero y colaborador que he tenido en mi vida, gracias por tu comprensión y tolerancia, así como tus enseñanzas, pero sobre todo gracias por ser un gran y leal amigo.

Al *Fis. Raymundo Hugo Rangel Gutierrez*, por transmitir su conocimiento y compartir su experiencia, por brindarnos la guía que necesitamos en los momentos más difíciles y por apoyarnosme en la culminación del mayor de mis esfuerzos, convertirme en Ingeniero.

A *Irma* Y *Gerardo*, amigos y compañeros de la carrera, con los cuales no solo comparti unas cuantas horas en el aula, sino unos de los momentos mas increibles de mi vida.

A *Matilde* Y *Fernando*, compañeros de trabajo, y más que eso amigos y mentores del verdadero desempeño profesional, modelo de profesionistas y seres humanos que he conocido.

A *Miguel*, hermano en la fé, amigo en la esperanza y una mano abierta en la desesperación, gracias por tus consejos y sobre todo por creer en la humanidad.

A *Rimet* Y *Lucero*, dos personas con alto grado de calidad humana, gracias por sus chistes, por sus consejos, por preocuparse por este amigo suyo, pero sobre todo gracias por tener la capacidad de ver más allá del hombre y creer en el profesionista.

Al *Grupo De Jovenes*, amigos incondicionales, una extensión de mi mismo y de mi familia, gracias por su apoyo y comprensión, pero sobre todo por mostrarme que Dios existe dentro de cada uno en diferentes formas.

A *Todos* Y *Cada Uno De Mis Amigos* Y *Compañeros*, quienes han compartido conmigo un poco de su existencia, conocimiento y de su frescura de vivir, gracias por los momentos de alegría y tristeza, por el esfuerzo y la calma, gracias por ayudar a crear una forma de ser autentica, autónoma y que sobre todo con sus experiencias aprendió a vivir.

A *Todos Mis Profesores*, desde mi maestra de primaria que me llevo la mano para poder escribir mis primeras letras, hasta mis maestros de la facultad, gracias por su esfuerzo, y por la transmisión de todo el conocimiento y experiencia adquiridos, pero ante todo gracias por formar el criterio y pensamiento del ser humano.

A *La Universidad*, no solo el campus, las aulas, o los elementos que le conforman, sino a toda una Institución con tradición, disciplina y virtud de unificar criterios y hacerlos escuchar, gracias por brindarme los elementos necesarios, para desarrollar los deseos de un joven estudiante ansioso de poder experimentar su propia conversión hacia una nueva forma de vivir: la Ingeniería.

Alfredo F.

Índice

Capítulo 3 Dispersión dinámica

Capítulo 4 Análisis y diseño

Capitulo 5 Implementación

Capitulo 6 Pruebas

Capítulo 7 Conclusiones

Capítulo 8 Bibliografía

Apéndice A Tabla de Códigos

Apéndice B Impulso ver. 1.0. Manual del Usuario

Apéndice C Código Fuente

Capítulo 1

Introducción

Los avances tecnológicos y los crecientes volúmenes de información, impulsan de manera gradual y constante la innovación de técnicas orientadas al manejo y procesamiento de la información.

Estas innovaciones se desarrollan por separado en dos grandes vertientes : la lógica y la física. La vertiente lógica manifiesta su labor por medio del desarrollo de estructuras de organización interna, capaces de llevar a cabo las acciones necesarias para el manejo de la información; mientras que la vertiente física realiza un trabajo más selectivo y cualitativo con lo relacionado al medio donde ha de almacenarse la información.

Los nuevos sistemas en cualquiera de sus dos vertientes sustentan su desempeño a través de una serie de herramientas, las cuales nos permiten de alguna manera lograr el procesamiento de la información. Sin embargo, los crecientes volúmenes de información superan por mucho a los sistemas antes citados. Pero aún así, el empleo de estas herramientas es consistente, destacando la computadora como una de las herramientas más populares en este tipo de lides.

Por lo que respecta a la computadora, las técnicas o procesos que utiliza en estas cuestiones se les conoce como estructuras de datos o de archivos, las cuales no se encuentran por sí solas en programas desarrollados o como paquetería específica; por el contrario, estas técnicas vienen incluidas en una gran variedad de programas, como pueden ser hojas de cálculo, procesadores de texto, empaquetadores de información, etcétera.

1.1 Estructuras de datos

Las estructuras de datos las podemos conceptualizar como aquellos esquemas de organización, los cuales nos permiten controlar determinada información así como también permiten la realización de diversas operaciones sobre los elementos componentes de la información, a estos elementos se les conoce como registros. Las estructuras de datos realizan el manejo de la información a través de procedimientos que pueden ser de una naturaleza estática, en el caso de que la estructura o esquema implementado, maneje una fuente de almacenamiento fija; o dinámicos, para los casos en los cuales la naturaleza de la aplicación requiera de obtener los recursos para el manejo de un tamaño indefinido dentro de la fuente de almacenamien-

to; o bien, en algunos casos se hace necesaria la combinación de ambos esquemas.

Algunos de los criterios observados para la implementación de las estructuras de datos de cualquier naturaleza, son los siguientes :

- ✓ **La implementación**, depende de la complejidad de las estructuras, se analiza la factibilidad de implementarse en cualquier lenguaje de programación existente;

- ✓ **La información**, es uno de los puntos más sobresalientes en el aspecto de la implementación, ya que dependiendo de la información que se tenga de esta, hará posible su construcción;

- ✓ **La popularidad** es otro de los puntos que nos permite apreciar el uso que tiene la técnica, así como de las variaciones que existen en torno a ella.

Como podemos observar, estos criterios no son los únicos, pero son los más utilizados para poder evaluar una técnica sobre otra.

En el universo de las estructuras de datos utilizadas para el manejo y recuperación de información, existe una gran variedad de técnicas y métodos. Por citar algunas de ellas, tenemos a las estructuras secuenciales, las secuenciales indexadas, los árboles binarios, los arboles B+, la dispersión, etcétera. Todas estas técnicas cumplen con un cometido especial: ser los instrumentos y mecanismos a través de los cuales se pueda ofrecer la información de una forma más ágil y eficaz.

1.2 Selección

Dentro de las estructuras anteriormente citadas, se encuentra la dispersión, la cual abarca uno de los métodos más eficaces para la implementación de algoritmos destinados a la recuperación de información. Esta se encuentra definida por los siguientes atributos :

- ✓ A diferencia de las otras técnicas, no requiere almacenamiento adicional para mantener un índice (aunque normalmente se requiere del manejo de algunas estructuras auxiliares de almacenamiento externo adicional);

- ✓ facilita y agiliza las operaciones de inserción y borrado de los registros, y

- ✓ permite recuperar un registro con muy pocos accesos a las unidades de almacenamiento externas.

Dentro de la misma dispersión existe una amplia gama de versiones. Esta variedad nace a partir de la constante búsqueda de la perfección dentro de la técnica, y es en este conjunto de opciones en donde encontramos a la dispersión estática, extendible, dinámica, lineal, virtual, etcétera. Estas variantes pueden ser encontradas ya sea en su forma básica o en una combinación con algún otro algoritmo que incluya procesos pertenecientes a las anteriores.

1.3 Ambientes

Un aspecto que es determinante en el diseño e implementación de las técnicas a utilizarse es el medio ambiente operativo o sistema operativo en el cual han de desenvolverse. Ya que dentro de las aplicaciones existen esquemas que se han llegado a desarrollar en diversas plataformas, y cada una de estas versiones, por así decirlo, puede ya no cumplir con la esencia principal del algoritmo, por lo que se altera su función completamente.

Una de las alternativas que se han intentado implementar para brindar una solución a este problema ha sido el establecer estándares en los lenguajes de programación. Un clásico ejemplo de lo anterior es el lenguaje de programación C/C++. Ya que el código que se desarrolle en una plataforma con estos lenguajes, será lo suficientemente compatible y transportable si se compila en otra plataforma.

1.4 Hacia adelante

El presente trabajo muestra los aspectos más esenciales de la dispersión, que es una de las es-

tructuras de datos más eficientes y más utilizadas en el manejo de archivos, y recuperación de la información. Así, el segundo capítulo nos muestra los aspectos más relevantes de la dispersión en su forma estática; mientras que el tercero hace lo propio con el aspecto dinámico. El cuarto capítulo nos muestra el análisis llevado a cabo para una nueva propuesta de dispersión; mientras que el quinto capítulo muestra los mecanismos y herramientas utilizados para la implementación del algoritmo en un lenguaje de programación. El sexto capítulo sirve para ilustrar el funcionamiento de la propuesta en una aplicación desarrollada bajo ambiente Windows. Y por último, el séptimo capítulo nos ofrece las conclusiones a las que se ha llegado con la realización y puesta en marcha de una nueva propuesta en el terreno de la dispersión dinámica, y por ende, en el campo de las estructuras de datos.

1.5 Referencias

[1] C. J. "Keith" van Rijsbergen. Four : File Structures. http://www.dcs.glasgow.ac.uk/Keith/Chapter.4/Ch.4.html. 1996.

[2] Data structure - PC Webopaedia Definition and Links. http://www.pcwebopaedia.com/data_structure.html. 1997.

En los últimos años, los esquemas que se habían estado utilizando para estructurar los grandes archivos de información, comenzaron a ser obsoletos y muy poco prácticos. Esto debido a las variantes y crecientes necesidades dentro del terreno del procesamiento de información, ya que estos se vieron prontamente envueltos dentro de una mezcla de conceptos y técnicas de dos áreas que inicialmente fueron vistas como vertientes con distintas aproximaciones, ambas con un objetivo común: el control de almacenamiento y el manejo de los datos para ser utilizados según los intereses del usuario. Considerando hasta este momento una optimización del espacio ocupado en memoria.

Una de las vertientes comprendía el uso de estructuras de datos apropiadas para llevar a cabo manejos e implementaciones de sistemas de información. Las cuales fueran capaces de dar soporte a las funciones de la memoria de control - en su momento se describirá con más detalle este término -, la cuál es la parte en donde se encuentra contenida la información. La otra aproximación se encaminaba con respecto a los métodos de acceso apropiados; de tal manera que se disminuyera la capacidad requerida por los dispositivos donde se lleva a cabo el almacenamiento.

Aunque ambas vertientes se desarrollaron paralelamente por algunos años, la idea del manejo de la memoria de control aunado a los métodos de acceso, dio a luz a una nueva tendencia que permitió el mayor aprovechamiento de los recursos de hardware, con los cuales se contaba hasta dicho momento y con los que se tienen en la actualidad. La tecnología retomó su camino y decidiendo no quedarse atrás, proporcionó una base más amplia, consistente y eficaz para llevar a cabo posteriores mejoras. De las técnicas y métodos mencionados con anterioridad, se desprende uno, el cual podemos considerarlo como uno de los métodos más simples y efectivos para organizar y almacenar los datos de información contenidos en archivos. A este método se le conoce como dispersión.

2.1 Fundamentos de la dispersión

La dispersión es un proceso que permite recuperar, insertar y borrar datos en un conjunto de información de manera bastante rápida, en esto reside su utilidad. Este proceso mapea llaves - valores únicos asignados a entidades - en índices de registros en un archivo, donde se encuen-

tra la información de interés de la entidad correspondiente.

En un proceso de dispersión intervienen tres objetos:

- ✓ La función de dispersión,
- ✓ la tabla de mapeo y
- ✓ el archivo donde se encuentra la información de interés.

La función de dispersión transforma o convierte una llave a un índice o posición en la tabla de mapeo, y de ahí se toma el número de registro en el que se encuentra la información de interés. Finalmente se recupera ese registro del archivo en que se encuentra. Una función de dispersión establece, en principio, una relación uno a uno entre la llave y la información de interés de una entidad, sin embargo, diferentes llaves pueden mapearse en un mismo índice de la tabla por razones diversas, a esto se le conoce como colisión. Estas se resuelven mediante diferentes estrategias

Los primeros esquemas de dispersión se realizaron con tablas de tamaño fijo, es decir, de naturaleza estática. Otras categorías de procesos de dispersión son los dinámicos, extendibles, etcétera.

Una llave es una cadena de caracteres numéricos o alfanuméricos, que se asocia con la información que se encuentra contenida en un registro dentro de un archivo. Por definición, una llave debe ser única y no ser nula. Lo cuál quiere decir, que debe de estar asociada con cada registro de información que se encuentre dentro del archivo, guardando siempre la relación uno a uno entre las llaves y la información existente.

Debido a que el rango de las llaves es normalmente mucho mayor que él número de elementos almacenados en las celdas. En ocasiones se llegan a presentar situaciones debido a que se desconoce el tamaño de la tabla, y a la capacidad de la función de transformación. Estos factores en conjunto hacen que la función provoque la doble asignación (colisión) de una localidad

dentro de los archivos - más adelante, daremos una visión más extensa de las colisiones, y algunas soluciones alternativas para resolverlas -.

El funcionamiento básico de esta técnica puede observase a partir del esquema planteado en la Figura 2.1, en la cual se muestra el proceso de dispersión: la llave de acceso K se transforma mediante la función de dispersión H, cuyo resultado permite hacer la consulta para recuperar y mostrar la información de interés.

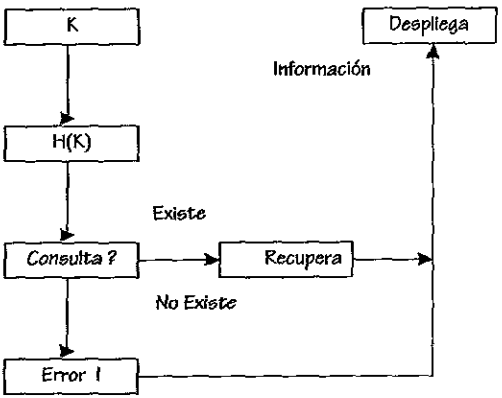


Fig. 2.1 Diagrama básico del proceso de dispersión.

La función de dispersión óptima es aquella que distribuye de manera uniforme las llaves en las entradas del arreglo (tabla o directorio), minimizando con ello las coincidencias o colisiones.

La frecuencia y distribución con que se presentan las colisiones, permite determinar si la función de transformación elegida es la óptima, ya que en caso contrario, debemos buscar una función idónea, que permita minimizar el riesgo de colisiones o coincidencias, y obtener una distribución mucho más uniforme en el directorio.

Para los casos en los cuales se presentan colisiones, las técnicas disponibles para llevar a cabo una solución a la presencia de esta situación, se plantean dentro del siguiente contexto. Ya que hemos entendido que se trabaja con una estructura que no es variable por naturaleza propia, se hace necesario modificar el tamaño de la tabla de dispersión, de tal manera que esto conlleva implícitamente rediseñar de la función de dispersión utilizada, para poder llevar a cabo la redispersión de todos los elementos que se hallaban contenidos dentro de la misma tabla.

Las técnicas de resolución de colisiones son distintas, dependiendo del enfoque desde el cuál

se desee abordar el problema. Dentro de las más importantes se cuenta el encadenamiento separado, el encadenamiento conjunto y el direccionamiento abierto. Enseguida veremos los detalles más importantes, y el funcionamiento básico de cada una de las anteriormente mencionadas.

El encadenamiento separado, es una de las técnicas más simples de comprender, en cuanto a funcionamiento, pues trabaja sobre las siguientes operaciones: insertar, borrar y buscar; ya que en contraste a las otras técnicas de manejo de colisiones, a esta se le puede considerar un híbrido entre lo referente al cálculo de la dirección y el procesamiento tipo lista, lo cuál proporciona dos grandes ventajas:

- ✓ Las bajas de los registros de información, no logran degradar el desempeño de la tabla de dispersión, no importando el número de direcciones que se encuentren disponibles dentro de esta.

- ✓ Impide el desbordamiento a causa de los problemas de almacenamiento suscitados por las colisiones.

En lo referente al encadenamiento conjunto, es aquella técnica que utiliza ligas, las cuales emanan de los bloques de registros que se llegan a desbordar, estos son almacenados dentro de la misma localidad y hacen referencia a otra celda dentro de la tabla de dispersión, colocada en una área separada a la del desbordamiento. Este método tiene la ventaja de que todo el espacio es completamente utilizado, con excepción de cuando se llegan a traslapar las ligas de información, que hacen referencia a otras localidades de la tabla.

Dentro del direccionamiento abierto se tienen características similares a las técnicas antes mencionadas, aunque conlleva implícitamente ciertas desventajas, ya que todos los casos tienen una alta posibilidad de llegar a presentarse, por lo que resulta que es una de las mejores técnicas para la resolución de colisiones. La intensión es almacenar el elemento de una localidad de memoria A_0, pero, si la función de transformación al momento de arrojar dicho valor encuentra que está ocupada, entonces busca la celda más cercana que se encuentre disponible y procede a descargar los datos de información.

2.2 Funciones de dispersión

Dentro de la terminología convencional, podemos decir que la dispersión se encuentra basada en el concepto de números aleatorios, es decir, la distribución de registros de información al azar. El principal propósito del uso de la teoría de números aleatorios, es la transformación de una distribución desconocida, sobre el dominio de K, en una distribución uniforme en un archivo.

Las funciones de dispersión son aquellas que proveen los mecanismos necesarios para poder manipular las tablas de dispersión. Esto es, que dada la llave para una entrada en particular, dicha función puede llevar a cabo los cálculos asociados al valor de la dirección, de una manera rápida y eficiente. Al contar con dicho valor asociado, permite llevar a cabo una búsqueda y recuperación de la información con un porcentaje bastante alto de confiabilidad. De lo anterior, nos podemos dar cuenta que existe al menos una entrada, con una llave dada a la tabla de dispersión, pero que sin embargo, esto no nos garantiza que esta sea distinta a cualquier otro, el cuál puede ser llevado a cabo en cualquier otro tipo de función.

En contraste a lo anterior podemos afirmar que el propósito primordial de una función de dispersión, es de alguna manera "romper" con todas aquellas regularidades que se encuentren presentes dentro del conjunto de información fuente, dándole una nueva distribución que permita mayores facilidades para el control de almacenamiento y recuperación.

Esto debido a que debemos de tener presente que la dispersión procede fundamentalmente en dos pasos:

- ✓ Llevar a cabo la transformación del elemento K, dentro de un número (X). En la mayoría de los casos (X) se maneja como número entero, que tenga relación con los elementos de almacenamiento disponibles dentro de una tabla de dispersión, aunque ocasionalmente en algunas aplicaciones se manejan como números reales $0 \leq (X) \leq 1$.

- ✓ Manipulación y cifrado de la llave K, para la obtención de H(K), en virtud de

que las técnicas de cifrado son sensibles a la longitud y definición del tamaño de la misma llave, y en función de este tipo de manejos dentro de las cifras dependerá la capacidad y efectividad de la función que se implemente para utilizarse dentro de alguna aplicación.

2.3 Criterios para elegir una función de dispersión

La mejor función de dispersión es aquella que tiene la capacidad de mapear N llaves, dentro de N celdas de almacenamiento contenidas dentro de una tabla de dispersión asociada, sin producir ninguna clase de sinónimos (colisiones).

En base a lo anterior los dos principales criterios para elegir a una función de dispersión, se encuentran basados en facilidad y rapidez dentro del cálculo del valor de dirección, así como de la capacidad de llegar a establecer una distribución homogénea de la información, a través de un intervalo de índices, los cuales hacen referencia a celdas de almacenamiento dentro de una tabla de dispersión, utilizada como directorio de control para la búsqueda y recuperación de archivos.

Si podemos conocer por anticipado exactamente las características de las llaves que se van a presentar, es posible tomar en cuenta los casos que nos presentaran dificultades, o que necesiten un seguimiento con mayor atención, de tal manera que es posible diseñar la función de dispersión apropiada, para el tipo de llaves y datos que nos encontremos manejando, de tal manera que nos proporcione la eficiencia y desempeño requeridos por la aplicación.

Una de las características que debemos de tomar en cuenta, es la generalidad del proceso llevado a cabo por la función de dispersión, el proceso más usual comprende la partición de la llave, las cuales pueden ser llevadas a cabo de diversas maneras, y de las cuales al final obtendremos un valor de dirección, qué será usado como un índice uniformemente distribuido sobre la tabla de dispersión correspondiente.

Es de este proceso donde proviene la palabra "dispersión", ya que esta secuencia de operaciones transforma a la llave en un valor de dirección, el cual ciertamente lleva un parecido en

poco o nulo grado, con la llave principal. Al mismo tiempo, se espera que los patrones que puedan presentar las llaves con respecto a la información, sean destruidos, de tal modo que las direcciones sean distribuidas tan aleatoriamente como sea posible.

Para ilustrar lo anterior, es necesario que veamos el funcionamiento de algunos de los métodos de diseño de funciones de dispersión. A continuación veremos cuales son los más utilizados dentro de la dispersión de tipo estático:

- **Truncamiento.** Esta técnica consiste en ignorar cierta parte de la llave, y usar los dígitos restantes de la misma como el valor asociado a las celdas de la tabla de dispersión (en el caso de utilizar campos alfanuméricos, estos deberán pasar por la correspondiente conversión a formato numérico). Si tuviéramos el caso, de llaves con una longitud de 8 dígitos, y una tabla asociada tiene disponibles alrededor de 1000 localidades, podemos tomar el primero, segundo y quinto dígitos, contando a partir de la derecha, los cuales podrán servir para llevar a cabo la función de transformación. Por ejemplo tenemos la llave 62538194, la función nos devolvería el número 394. El truncamiento tiene la ventaja de ser muy veloz, pero con frecuencia no logra hacer una distribución de las llaves de manera uniforme a lo largo de la tabla.

- **Doblamiento.** Este consiste en seccionar la llave en varias partes, luego estas se combinan de manera adecuada, con lo que se obtiene un valor asociado a la dirección de la tabla de dispersión. Por ejemplo, si llegáramos a tener una llave de 8 dígitos, esta puede dividirse en grupos de 3, 3 y 2 y estos subconjuntos obtenidos de la llave podrían sumarse, si fuera necesario; para obtener el valor de direccionamiento, el número obtenido se trunca al número de dígitos que maneja la tabla de dispersión. Entonces la llave 62538194, se separaría de la siguiente manera: $62 + 538 + 124 = 724$, que sería la localidad de almacenamiento a la que se encuentra dirigida, con lo que resolveríamos el problema mediante este método.

- **Aritmética modular.** Se procede inicialmente a convertir la llave en un número entero, y una vez que esta operación se ha efectuado, el valor obtenido se divide entre el tamaño de la tabla de dispersión asociada, pero en este caso el resultado que a nosotros nos

importa para llevar a cabo este tipo de distribución es el residuo. Este tipo de función de dispersión realizada al tomar un residuo, depende en sumo grado del tamaño de la tabla asociada, ya que si se tiene una tabla de tamaño muy pequeño, la mayor parte de las llaves tienden a ser mapeadas, dentro de la misma localidad de almacenamiento, en tanto que la parte sobrante permanece sin llegarse a usar. Por esta razón en particular se hace preciso el uso de un tamaño de la tabla de dispersión, el cuál sea un número primo, ya que esto permite a la función de dispersión, tener el menor porcentaje de ocurrencia de colisiones. Por ejemplo, si tenemos una tabla con capacidad para el almacenamiento de 1000 registros, lo más conveniente en ese caso es aproximarla a 1009 ó bajarla a 979, ya que esta característica nos permitirá que la función de transformación lleve a cabo un mayor aprovechamiento del espacio dentro de la tabla de dispersión asociada.

2.4 Diseño de una función de dispersión

En la mayor parte de las aplicaciones, el hecho de utilizar una función de dispersión ordinaria, no es suficiente para darnos las características de confiabilidad que nosotros esperamos, ya que con lo visto anteriormente, podemos darnos cuenta que no existe una función de transformación que nos garantice al 100 % la omisión de los sinónimos. Lo cuál en la etapa de diseño de un sistema, es de vital importancia para el correcto funcionamiento del mismo, así como la entera satisfacción del cliente al poder ver su información dentro de la pantalla y poder hacer los ajustes necesarios a su base de información.

De esto surge la necesidad, de ya no estar al margen del funcionamiento de la función misma, sino de darle un nuevo enfoque que nos permita en este caso, tener la capacidad de llevar a cabo una distribución totalmente uniforme, sin tener la menor expectativa de que se pueda llevar a cabo la temida colisión de información. El "temor" de muchos programadores se hace patente en esta parte, y la frase de "todo lo bien hecho puede llegar a ser funcional", se descarta de antemano, ya que si nada de lo que existe a tu alrededor funciona, desarrolla lo que vaya de acuerdo a las necesidades que deseas satisfacer.

Con el propósito de tener la ventaja de llevar a cabo una distribución mucho más uniforme, a las funciones de dispersión se les puede otorgar la capacidad de involucrarse dentro de aspec-

tos cada vez más complicados, hasta ser lo más controversial posible, de tal manera que estas puedan ser abarcadas desde conceptos muy sencillos, como pueden ser sumas, restas y conversiones de caracteres, hasta llegar a implementar funciones que se encuentren basadas en cálculos asociados, a los valores de dirección en que se encuentran almacenadas dentro de la memoria real del equipo.

En el anterior apartado vimos funciones muy simples, para poder llevar a cabo una dispersión de información en una tabla, ahora presentamos un ejemplo de una función diseñada, la cual es bastante eficiente en su desempeño, con un grado bastante alto de confiabilidad.

Siendo esta la función, la cual se encuentra codificada en el lenguaje de programación C:

```
static int hash(char *K,int nbits) {
register int n=0,loop,len = nbits;
register char *str = K;
#define HASHC  n = 65599*n + *str++
#define stop       7

 if(len > 0 ) {
   loop = (len + stop) >> 3;
   switch(len & stop) {
      case 0 : do {
                    HASHC;
                    case 7 : HASHC;
                    case 6 : HASHC;
                    case 5 : HASHC;
                    case 4 : HASHC;
                    case 3 : HASHC;
                    case 2 : HASHC;
                    case 1 : HASHC;
                }while(--loop);
      }
   }
   return n;
}
```

2.5 Referencias

[1] Bertizz, A.T. Data Structures Theory and Practice. Computer Science and Applied Mathematics. Academic Press. Inc. New York, New York.1975.

[2] Kruse, Robert L. Estructuras de datos y Diseño de programas. Prentice Hall Hispanoamericana. México. 1989.

[3] Sprugnoli, Renzo. Perfect hashing Functions : A single Probe Retrieving Method for Static Sets. Communications of the ACM. Vol. 20; No. 11; November 1977. New York, New Jersey. Pags 841-850.

[4] Collins, William J. Data Structures and Object Oriented Approach. Addison Wesley Reading, Mass. 1982.

[5] Samet, Hanan. The Design and Analysis of Spatial Data Structures. Addison Wesley Reading, Mass. 1978.

[6] Nievergelt, Jurg and Klaus H. Hinrichs. Algorithms and Data Structures with Applications to Graphics and Geometry. Prentice Hall, New York. 1979.

[7] Cichelli, Richard J.. Minimal Perfect Hash Functions made simple. Communications of the ACM. Vol 23; No. 12; December 1981. New York, New Jersey. Pags. 17-19.

[8] Jaeschke, G; Oesterburg, G. Technical Correspondence on Cichelli's Minimal Perfect Hash Functions Made Simple. Communications of the ACM. Vol 23; No. 12 December 1980. New York, New Jersey. Pags 728-729.

[9] Jaeschke, G. Reciprocal hashing : A Method for Generating Minimal Perfect hashing Functions. Communications of the ACM. Vol 24; No. 12; December 1981. New York, New Jersey. Pags. 829-833.

[10] Severance, Dennis & Duhne Ricardo. Practitioner's Guide to Addresssing Algorithms. Communications of the ACM. Vol. 19; No. 6; June 1976. New York, New Jersey. Pags. 316-324.

Capítulo 3

Dispersión dinámica

Como se puede observar al leer el capítulo anterior, el acceso directo de archivos por medio de la dispersión es de lo más eficiente para recobrar los registros de un archivo a través de una llave. Pero tiene un gran incoveniente: la tabla es tamaño fija, pues es una estructura estática, la cual resulta poco eficiente en aplicaciones donde el número de registros no se conoce de antemano o donde el número de los mismos fluctúa ampliamente.

Para estos casos, es más conveniente contar con una estructura dinámica para la tabla, en la cual se realizan las expansiones y contracciones automáticamente dictadas por el número de registros en el archivo. Especificamente, una tabla con estructura dinámica debe poseer tres propiedades, que son:

- ✓ La tabla deberá expandirse automaticamente para acomodar a los nuevos registros.

- ✓ La expansión no requerirá una nueva organización de la tabla o de la reubicación de los registros en el archivo.

- ✓ La tabla deberá contraerse automáticamente cuando la cantidad de registros en una página, caiga por debajo de un valor predeterminado.

A la última consideración se le conoce también como factor de carga. Al igual que en la expansión, la contracción no requerirá de una nueva organización de la tabla o la reubicación de los registros. La estructura del archivo deberá permitir la recuperación de un registro por su llave primaria con un solo acceso a la memoria secundaria.

3.1 Definiciones y terminología

Para explicar con más presición la dispersión dinámica, se defienen los siguientes términos:

- ✓ "H" denota a una función de dispersión dada.

- ✓ Cada registro será recobrado a través de uno de sus campos, siendo conocido como llave primaria y la denotaremos como "K".

- ✓ Un archivo "F", el cual es una colección de registros.

- ✓ Una página se define como un bloque de "b" registros, el cual se accesa mediante una operación de lectura.

- ✓ El espacio de utilización se define por la siguiente expresión:

$$\frac{n}{m \times b} \qquad \textit{Ecuación 3.1}$$

donde "n" es el número de registros en el archivo, "m" es el número de páginas usadas y "b" es la capacidad de la página.

- ✓ El factor de carga es aquel que contempla los valores máximo y mínimo de registros que le da soporte a una página (típicamente estos valores oscilan entre el 50% y 100% respectivamente).

- ✓ Si más de "b" registros son mapeados a la misma página, ocurre un desbordamiento, donde debemos tener presente que "b" es la capacidad de la página y por lo regular se le considera como el valor del factor de carga máximo.

- ✓ La dispersión dinámica es una combinación de las técnicas de dispersión con la estructura de árbol, conocida como "tries".

3.2 Tries, la base para la dispersión dinámica

Un "Trie" es un árbol del tipo binario, en el cuál la ramificación se determina por una secuencia de bits. La secuencia se obtiene de la siguiente forma: primero, se mapea K dentro de la función de dispersión. Segundo, una vez obtenido el valor de dispersión, la secuencia se establece en base al número de registros contabilizados hasta el momento de ingresar uno nuevo.

En este estudio consideraremos que el valor devuelto por H(K) será un número binario compuesto por "n" bits. La asignación de los bits puede eligirse en cualquier orden, pero por simplicidad nosotros asignaremos que el nivel "i" estará determinado por el i-ésimo bit más significativo. Como un ejemplo considere el trie de la siguiente Figura 3.1, el cual direcciona 3 páginas A, B y C.

Los registros que comienzan con la secuencia de bits 00... serán colocados en la página A, aquellos cuya secuencia de bits sea 01... serán colocados en la página B, y aquellos cuya secuencia de bits comience con 10... serán colocados en la página C. Si la página A se satura y se desborda, se dividirá en dos páginas, como se muestra a continuación en la Figura 3.2

En este caso se hace necesario un bit adicional para dividir los registros que se encontraban en A, entre A y la nueva página D. Ahora la página A contiene aquellos registros que comienzan con la secuencia de bits 000..., y los correspondientes a la secuencia 001... serán mapeados a la página D.

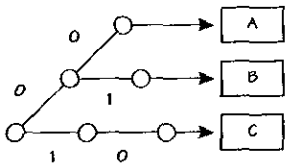


Fig. 3.1 Trie básico.

Como podemos ver, cuando se inserta un registro R_j, y con llave K_j; primero localizamos el nodo raíz el árbol de búsqueda ó trie, obteniendo $H(K_j)$. Entonces verificamos el camino o ruta a desarrollar, según lo indiquen la secuencia de bits que le sigan, hasta que podamos alcanzar un nodo terminal; este nodo contendrá un apuntador el cual señala a la siguiente página siguiente en el cuál el registro va a ser almacenado.

Si la página se encuentra lleno ó muy próximo a llenarse, el nodo del árbol dentro del cual se encuentre la información se divide en dos nuevos nodos y una nueva página es asignada a uno de los anteriores y los registros se dividen entre las páginas que aparecen debido a esta división. Inicialmente un trie consta de un solo nodo, al cuál le denominamos

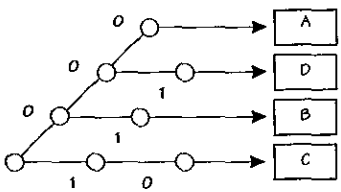


Fig. 3.2 Trie expandido.

nodo terminal, el que apunta a uno de los "m" páginas inicialmente existentes. El que un registro se almacene en una página u otra se determina solo por la correspondiente secuencia binaria que le corresponde al mismo. Lo que se busca como resultado a partir de lo anterior es la generación de arboles de dispersión aleatoriamente balanceados, es decir, pretendemos que los registros almacenados en los subárboles izquierdo y derecho de cualquier nodo puedan ser iguales. Este tipo de árboles nos proporciona la ventaja de tener menores rutas en longitud y mejor utilización del espacio de almacenamiento.

La búsqueda en un árbol de dispersión podrá optimizarse si logramos contraer la estructura dentro de un directorio. Este directorio puede entonces ser direccionado por una función de dispersión. Si la función elegida es uniforme (distribuye de manera aleatoria las llaves sobre el rango de la función), entonces podemos suponer que el trie utilizado es balanceado. La forma de realizar la asignación de un trie a un directorio se ilustra en la Figura 3.3.

La profundidad del trie de la Figura 3.1 es de 2, debido a que se requirieron dos bits, para llevar a cabo la distribución de los registros dentro de las páginas. El segundo bit, sin embargo, no fue utilizado para la página C, pero de todas maneras se muestra para ilustrar como se forma el directorio completo.

Una vez que el trie se ha definido completamente, como el de la Figura 3.2, los bits utilizados como el camino de búsqueda a través del árbol, son ahora los índices mediante los cuales se obtiene el acceso al directorio que controla al trie colapsado. Como es posible de apreciar en la Figura 3.3, los primeros bits de las llaves procesadas son ahora un índice dentro del directorio; existen, sin embargo, entradas redundantes al directorio, como lo son las dos entradas al directorio usadas para la página C. Este problema de entradas redundantes, depende directamente, de la uniformidad de la función de dispersión elegida dentro de la aplicación a desarrollar.

Ahora consideremos la situación dentro de la cuál la página A se satura, y termina por desbordarse, como consecuencia se agrega una página D, el trie se muestra en la Figura 3.4A y su correspondiente transformación en un directorio se muestra en la Figura 3.4B.

Podemos notar de lo anterior que un bit extra fue requerido para llevar una clara diferencia en-

tre la página A y el D, necesitando 3 bits, y un árbol con tres niveles.

De estos tres bits se deriva un directorio capaz de direccionar 8 entradas ($2^3 = 8$). De estas entradas muchas de las mismas son redundantes, para ello observemos como la página C tiene 4 entradas distintas asignadas por el directorio. En consecuencia, esta situación en lo particular nos puede acarrear un nivel de "n" direcciones que incremente por consecuencia el tiempo de acceso.

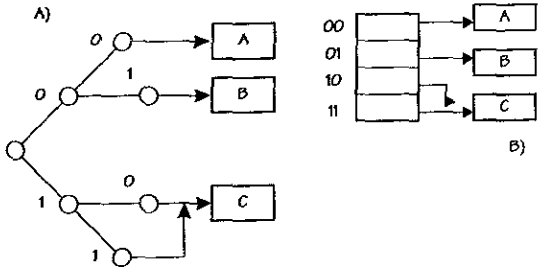


Fig. 3.3 Trie colapsado en un directorio.

Una forma diferente de colapsar un trie, se muestra a partir de la Figura esquemática 3.5, la cual consiste básicamente en la eliminación total del directorio a partir del cual se encuentran las páginas.

Este directorio puede ser eliminado siempre y cuando se mantengan a las páginas que contiene a dicha información dentro de espacios de almacenamiento contiguos dentro de la memoria; en el momento en que el directorio se elimina, juntamente con este son eliminados los apuntadores de estructura del mismo, es decir las ligas de acceso indirecto hacia las páginas.

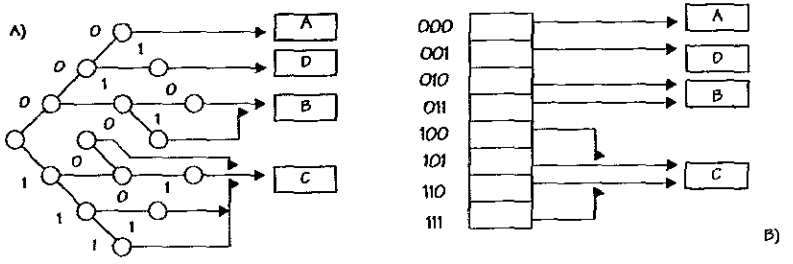


Fig. 3.4 Trie y Directorio.

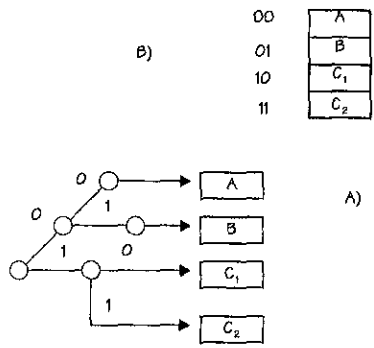


Fig. 3.5 Colapsado sin directorio.

Si consideramos a la página C que aparece en la Figura 3.3, podemos ver que los apuntadores a esta página, son aquellos que reproducen los niveles considerados como "bajos" de la estructura del trie. Una vez que el directorio se ha eliminado, el camino de búsqueda del árbol se transforma en un direccionamiento directo a la página que contiene la información buscada.

Dicho método de direccionamiento, requiere que existan tantas páginas disponibles en espacios contiguos de memoria, como entradas se encuentren en el directorio a eliminar. Como se puede observar en la Figura 3.5, el contenido de la página C se divide en dos páginas; lo que indica que a partir de la transformación del trie, el directorio no se hace absolutamente necesario. Los dos bits que previamente servían como índice al directorio, ahora se usan para dar la dirección directa de las páginas que contienen la información requerida. (Fig. 3.5 B).

3.3 Esquemas básicos de dispersión

Los sistemas de dispersión dinámica entran dentro de dos grandes categorías:

- ✓ Esquemas de directorio ó de dispersión extendible y/ó
- ✓ Esquemas sin directorio ó dispersión lineal.

Ambos esquemas se derivan del tipo de transformación usado para colapsar los árboles como los mostrados dentro del apartado pasado.

3.3.1 Dispersión extendible

La secuencia de pasos que componen a la dispersión extendible se muestra a continuación

dentro del siguiente diagrama de bloques:

- ✓ La primera transformación usada dentro de la dispersión extendible es muy similar a la función de dispersión utilizada dentro de los archivos de acceso directo. Las mismas propiedades que se mencionan en el capítulo anterior deben de ser contempladas para el diseño de esta función de dispersión a utilizarse.

- ✓ Se debe de llevar a cabo una distribución uniforme y aleatoria de las llaves sobre el rango de la función.

- ✓ Pequeñas modificaciones dentro de la llave utilizada, darán como resultado grandes variaciones con los valores obtenidos a partir de la función de dispersión ya implementada.

La segunda transformación extrae un cierto número de dígitos de H(K); existen varios criterios para la elección de estos dígitos. Convencionalmente se llegan a seleccionar los dígitos o bits más significativos del valor obtenido dentro de la función, mediante la cual se llevo a cabo la transformación, aunque a pesar de esto, los dígitos menos significativos o cualquier otro conjunto de los mismos pueden resultar igualmente efectivos.

Los dígitos extraídos de H(K) son usados como índice dentro de un arreglo unidimensional de apuntadores (a este se le denomina directorio). Este arreglo puede llegar a contener 2^d entradas como máximo, es decir, una para cada combinación de "d" dígitos extraídos a partir de H(K).

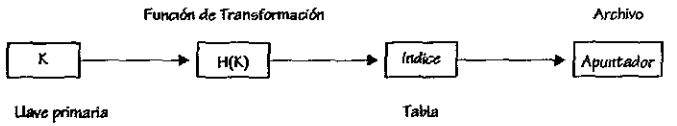


Fig. 3.6 Esquema representativo.

En la Figura 3.7 se observa que son utilizados los tres primeros bits de H(K) seccionando así el espacio de direccionamiento en 8 segmentos de igual proporción dentro del directorio.

Por ejemplo, supongamos que a partir de la función de dispersión obtenemos un valor como el siguiente: H(K) =01 1001 0110 0101; los primeros tres bits más significativos son 011, corresponden a la entrada con el índice del mismo valor, dentro del directorio, el cual almacena un apuntador que señala a la página C.

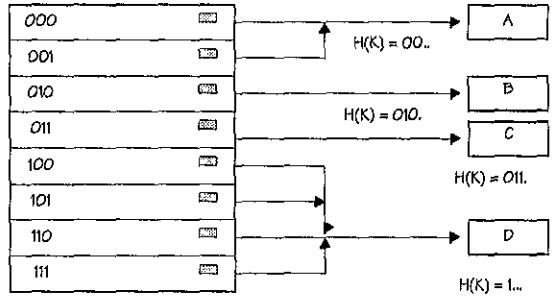


Fig.3.7 Directorio de orden 3 con 4 digitos.

Primero la llave se dispersa para obtener H(K). Segundo, los primeros "d" dígitos (bits) son extraídos para llevar a cabo la formación de un índice dentro del directorio. Tercero, el índice se usa para localizar dentro del directorio, la página en la cual se localiza la información deseada. Cuarto, este apuntador ayuda a recuperar la información requerida hacia la memoria principal; Quinto la información deseada contenida dentro del registro, se localiza en la página correspondiente.

3.3.1.1 Expansión y contracción de la tabla

Como ya se ha mencionado a lo largo de este capítulo, uno de los principales propósitos de la dispersión dinámica, estriba en permitir de manera flexible la expansión y contracción de un archivo, según se necesite para dar satisfacción a las necesidades de almacenamiento de información.

Para realizar el control de la expansión y contracción de las páginas que contienen la información, se establecen dos valores relacionados con la capacidad de las páginas, a estos valores se les conoce con el nombre de factor de carga mínimo y máximo; típicamente este factor es de

50 % y 100 % respectivamente. Así, cada vez que uno de estos valores es violado mediante la adición ó eliminación de un cierto registro se realiza un ajuste dentro de la estructura del archivo.

Considérese la estructura de directorio de la Figura 3.7, y supóngase que se llegará a realizar la adición de un nuevo registro de información, posterior a ser mapeado mediante la correspondiente función, y será agregado dentro de la página "D". Si éste se encuentra lleno o muy próximo a llenarse, y este no cuenta con el espacio suficiente para dar albergue al nuevo registro; entonces el archivo se expande.

Es entonces cuando se agrega una página nueva "E", dentro de la estructura de la tabla; la mitad de los apuntadores que se encontraban en dirección a la página "D" se cambian para señalar hacia la página "E", de la misma manera que se equilibraron las ligas, se lleva a cabo la misma distribución de los registros de información, dentro de las páginas disponibles. Estas ahora se encuentran a una capacidad mediana de ocupación con capacidad de recepción de un número amplio de registros, para realizar una nueva inserción.

El resultado de esta división se muestra a través de la Figura 3.8. De lo anterior podemos analizar que en el paso anterior a la división, todos los registros para los cuales H(K)=1... se encontraban en la página "D"; posterior a la división, ahora quedan de la siguiente forma: aquellos registros donde H(K)=10... se encuentran contenidos en la página "D" y aquellos donde H(K)=11... se encuentran en la página "E". El parámetro "d" que se muestra en cada página de la Figura 3.8 indica el número de dígitos de H(K) cuyo valor es común a todos los registros en la página.

Es importante mencionar que el valor de "d" debe de ser menor o igual que el número de dígitos "d", usado como índice dentro del directorio. El número de apuntadores que señalan a una página dada es siempre de $2^{(d-d')}$. Algunos autores denominan a "d" como profundidad global y a d' como profundidad local.
La división de páginas puede continuar según el archivo crezca, siempre que existan al menos dos apuntadores que se encuentren direccionados hacia la página que va a ser dividida.

Si existe una página para la cuál sólo hay un apuntador en el directorio, es decir que d = d' , y que se encuentre a punto de sufrir un debordamiento de información, el directorio deberá de ampliarse o doblarse de tamaño antes de que la página pueda ser dividida.

En otras palabras : Cuando dentro de la página el valor de la profundidad local es igual al valor de la profundidad global d = d', y la capacidad de la página que nos encontramos utilizando se llega a saturar y a sufrir un desbordamiento, el tamaño del arreglo se debe de doblar, y se hace necesario consecuentemente incrementar un bit más y juntamente con lo anterior, se deberá incrementar el número de entradas al directorio, ya doblado en tamaño.

Por ejemplo si asumimos que la primera de las tres anteriores condiciones se cumple, las páginas A y B en la Figura 3.9 podemos observar que estos no pueden mezclarse, ya que tiene diferentes valores de profundidad local. Las páginas A y D no pueden mezclarse porque sus registros no comparten los mismos valores dentro de los $d'-1$ dígitos de H(K). Sin embargo, las páginas D y E cumplen todas las condiciones y por lo tanto no es posible mezclarse.

El directorio debe de contraerse siempre que todos los pares de apuntadores tengan el mismo valor. Por ejemplo, un directorio de apuntadores : A, A, A, A, B, B, C, C, D, D, D, D, E, E, F y F debe de contraerse para producir un nuevo directorio con la mitad de todas las entradas. El valor de cada par de apuntadores dentro del antiguo directorio , es usado como un valor de referencia dentro del nuevo directorio, teniendo así los valores de apuntadores A, A, B, C, D, D, E, F. Por lo tanto un directorio con los valores de apuntadores A, A, A, A, B, B, C, C, D, D, D, D, E, E, E, E no es posible de contraerse, ya que no se pueden formar pares de parejas (debido a que no se tiene un número par) dentro de los valores de A y B. Podemos decirlo de otra manera, el directorio es posi-

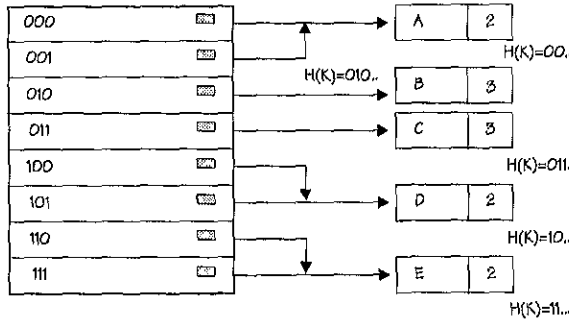


Fig. 3.8 Distribución de las llaves entre páginas.

ble de contraerse cuando la profundidad local d' de todas la páginas es menor que la profundidad global d.

El número promedio de páginas y entradas al directorio después de varias inserciones y desapariciones aleatorias de registros, se establecen a través de las siguientes fórmulas:

1.- Número de páginas.

$$\frac{N}{B \times \ln(2)} \qquad \textit{Ecuación 3.2}$$

2.- Número de entrada al directorio.

$$\frac{N}{B \times \ln(2)^2} \qquad \textit{Ecuación 3.3}$$

donde :

- ✓ *N es el Número de registros en el archivo.*
- ✓ *B es el Número de registros que pueden almacenarse en una página (capacidad de la página).*

Debemos de observar que la ecuación 3.3 representa básicamente un valor promedio. El número actual de entradas en cualquier tiempo será siempre una potencia de 2.

El directorio siempre contendrá un número mucho mayor de apuntadores, que de páginas, ya que algunos de estos últimos tendrán más de un apuntador. La razón o cociente promedio de apuntadores para una página es 1/ln(2) ó 1.44. Entonces el factor de carga promedio se deriva a partir de la fórmula 3.2 lo cual nos arroja un resultado del 69 %, por lo tanto el número de apuntadores estará cercano a $(2.08 \times N)/B$.

3.3.1.2 Estructuras de las páginas

Toda la teoría referente a la dispersión dinámica se ha enfocado al concepto relativo a la localización y búsqueda de la página, dentro del cuál se encuentra el registro de información que ha sido requisitado. Aún no se ha mencionado nada acerca de como llevar a cabo la localización del registro en la propia página. Esto se basa en dos aspectos importantes: Primero, el método que se ha usado para organizar la página internamente, no afecta de manera alguna el número de operaciones físicas de entrada / salida. Segunda, este tipo de organización no tiene un impacto de alta relevancia, dentro de las operaciones propias para llevar a cabo la recuperación de la información contenida, dentro el registro deseado para ser recuperado.

La estructura mantiene una simple organización cronológica - secuencial, lo cual con el mapeo de un bit es factible. Para encontrar un registro de información, cada combinación dentro de la página es examinada en secuencia, y ambas llaves se comparan, hasta el momento en que ambas compaginen o se encuentre el final de la página en la que nos encontremos buscando.

3.3.2 Dispersión lineal

El otro esquema básico en la dispersión dinámica, es el esquema "sin directorio" (directoryless scheme). Para este esquema se describirá el mapeo lógico asumiendo que el archivo está almacenado en espacios de dirección (de almacenamiento) contiguos lógicamente, como se muestra en la figura 3.10.

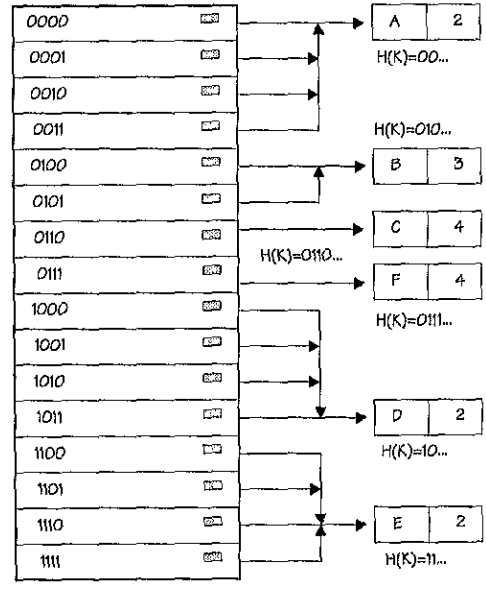


Fig. 3.9 Distribución de llaves de orden 4.

En la figura 3.10A el archivo esta al comienzo de la segunda de expansión $d = 2$, entonces el archivo contiene $2^d = 4$ páginas de datos primarios. La característica prin-

cipal de este esquema es que el directorio se elimina y H(Key) sirve para direccionar la página de datos primarios deseada.

La expansión en este es similar que en el esquema anterior. Cuando ocurre un desbordamiento en una página, se dispara una división de páginas, no es necesariamente la que se divide. En este caso, las páginas se dividenen orden secuencial sin estimar si la página dividida es la única que actualmente se ha saturado. En este esquema cierto número de digitos binarios o bits de H(Key) direccionan directamente a las páginas; en donde debe haber una página para cada valor de la función de dispersión, al igual que en el esquema del directorio había una entrada al directorio para cada valor de la función.

Si se debe ejecutar una expansión dentro del esquema de dispersión dinámica sin directorio, la solución idonéa es adicionar las páginas, una a la vez en un orden secuencial. A continuación veremos la descripción de un ejemplo que nos permitirá ilustrar la operación de este esquema de dispersión dinámica.

La Figura 3.10 nos muestra una expansión, dentro de un esquema de dispersión sin directorio. Comenzamos con las 4 páginas que se muestran dentro de la Figura 3.10A, esto nos dice que $d = 2$, por lo que requerimos de 2 bits para direccionar tales páginas. En este tipo de ejemplo el archivo se expandirá siempre y cuando cualquiera de las páginas contenidas dentro del mismo se sature.

Tenemos que la página b es la primer página que se satura. El archivo se expandirá con la adición de la página A, y realizando la división de los registros correspondientes, en las páginas 'a' y 'A', pero antes de esto, para poder diferenciar entre los registros de la página 'a' y los de 'A', es necesario que se añada un bit extra a la dispersión. Por lo tanto 2 bits se utilizan para direccionar las páginas 'b', 'c', y 'd'; mientras que para las páginas 'a' y 'A' se utilizan tres bits. Esto se encuentra ilustrado en la Figura 3.10B.
Puesto que el orden en que se encuentran las páginas ubicadas, obedece a un orden secuencial, la página 'a' se dividió mucho antes que la página 'b', siendo esta última la que llegó a saturarse, siendo necesaria la aparición de una página de desbordamiento (w) para albergar lo que se encontraba almacenado dentro de la página 'b'.

Posterior a un determinado número de inserciones, podemos ver que la página 'd' es la siguiente en el orden de saturación. (Figura 3.10B). Antes de que la página 'b' llegue a dividirse, una página B es agregada dentro de la estructura del archivo, por lo que los registros de la página 'b', y los correspondientes a la página de desbordamiento (w) se dividen entre ambas páginas que se encuentran disponibles ('b' y 'B'). Dos bits se usan para direccionar las páginas 'c' y 'd', pero se encuentran utilizándose tres bits para direccionar a las páginas 'a', 'b', 'A' y 'B'. Asumiendo que se lleven a cabo más inserciones, ahora tenemos que se satura la página 'd', agregándose entonces una página 'C', además de que la propia página 'c' se dividirá tal y como se ilustra dentro de la Figura 3.10C.

Para finalizar, tenemos que la página 'B' se llega a saturar, causando en consecuencia la adición de una página 'D' y sus páginas de desbordamiento 'X' y 'Y' se dividen, tal y como se muestra en la Figura 3.10D.

Ahora todas las páginas, requieren de al menos tres bits para lograr su direccionamiento; con lo que se ha completado la segunda expansión, y se vislumbra el comienzo de la tercera expansión, la cuál si se realiza, se dará de la misma manera. Por consiguiente la contracción del archivo se realiza de manera análoga, solo que a la inversa. Si tenemos un archivo, el cuál comienza con la n-ésima expansión, este contendra $s = 2^d$ páginas primarias, las cuales serán

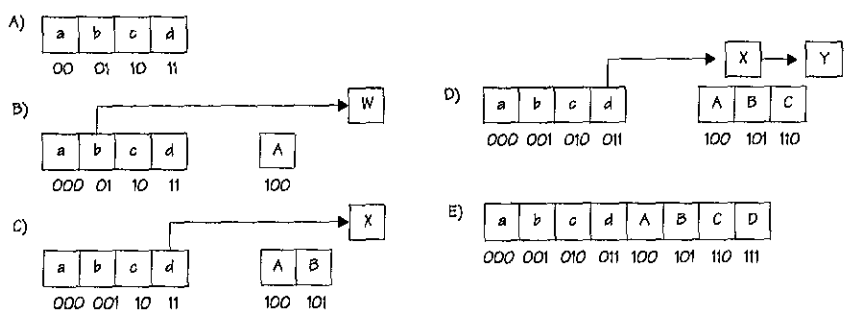


Fig. 3.10 Esquema de dispersión sin directorio.

direccionadas por $H_d(K)$ bits. Conforme más registros se lleguen a insertar al archivo, nuevas páginas serán agregadas al final del mismo. Para poder determinar cuál página será dividida con una nueva página agregada, se utiliza un apuntador "p".

Al comienzo de la n-ésima expansión (expansión d), el apuntador p señalará a una página 0 del archivo. Cuando otra página primaria se agrega al archivo, la página 0 (señalada por p) se dividirá colocando los registros en las páginas 0, y en la nueva página. Si existen $s = 2^d$ páginas, la nueva página agregada será la página s+p. Esta página p al dividirse, reparte los registros entre las páginas p y s+p, usando los $H_d(K)$ bits de H(K) para llevar a cabo el direccionamiento de los anteriores.

El apuntador p se incrementa entonces en 1. Cuando este alcance a s, entonces tendremos 2^{d+1} páginas dentro de la estructura del archivo; por lo que se incrementará a d, y se regresará al apuntador p al principio del archivo con el fin de comenzar la siguiente expansión.

Podemos observar que este esquema de dispersión, utiliza los primeros $H_d(K)$ y $H_{d+1}(K)$ bits de H(K) para llevar a cabo el acceso a las páginas de datos primarios. El procedimiento que se realiza es el siguiente:

```
Incio
   Si es necesario, detecta la cadena de desbordamiento
   O bien página := Hd+1(K);
Fin.
```

3.4 Utilización del espacio de almacenamiento

Teniendo en cuenta la ecuación 3.1 donde los "n" registros, las "m" páginas de datos, y los "b" registros por cada página, la utilización del espacio de almacenamiento es el cociente del número de registros, sobre el espacio reservado para el almacenamiento de los mismos.
Si no se tiene una o más páginas de desbordamiento, dentro del esquema de la dispersión dinámica, podemos esperar que la utilización del espacio de almacenamiento permanezca cercana o por debajo del 69 %.

Siempre que una página se llena, es decir que sobrepasa su factor de carga, esta se divide a su

vez en dos páginas, las cuales presentan una mediana capacidad. Con "b" registros, lo equivalente a una página llena, esta se divide y reparte sobre dos páginas nuevas, las cuales tienen una capacidad total de $2 \times b$; la utilización de las nuevas páginas es:

$$b/2b = 1/2 = 0.5 = 50\% \qquad \textit{Ecuación 3.4}$$

Después de cierto número de divisiones esperamos que la utilización sea mayor del 50 % pero menor del 100 %. Así con el tiempo la utilización del espacio de almacenamiento de los esquemas de dispersión será aproximadamente de ln(2) = 0.69 = 69 % , con lo que podemos esperar que se tendrán a la larga divisiones sucesivas en la mitad de las páginas.

Esta utilización del espacio de almacenamiento del 69 % se deriva del análisis del trie asociado con dispersión dinámica y una distribución uniforme de la llave.

3.5 Referencias

[1] Fagin R.; J. Nievergelt; N. Pipeenger & H. R. Strong. extendible hashing - a fast access method for dynamic files-. ACM Transactions on database systems. Vol. 4; No. 3; September 1976. New York, New York. Pags. 315-344.

[2] Larson, P. Dynamic hashing. BIT18; 1978. Pags.184-201.

[3] Aaron M. Tenenbaum; Yedidyab Langsam & Moshe J. A. Data structures using C. Pretince Hall, Englewood Cliffs, N. J.. 1990.

[4] R. J. Enbody & H. C. Du. Dynamic hashing schemes. Computing Surveys ACM . Vol. 20; No. 2. 1988. New York, New York. Pags. 85-113.

[5] Scholl, M. New file organizations based on dynamic hashing. ACM Transactions on databse systems; Vol. 9. No. 1. March 1981. New york, New York. Pags. 194-211.

[6] Severance, D. & Duhne, R. A practitioner's guide to addresing algorithms.:

Communications of the ACM. Vol. 19. No. 6. June 1976. New York, New York. Pags. 314-326.

[7] Budd, Timothy A. Classic data structures in C++. Addison-Wesley. Reading, Mass.. 1994.

Capítulo 4

Análisis y diseño

El desarrollo de un eficiente algoritmo de búsqueda ha sido en los últimos años una área alternativa de investigación en el campo de las ciencias de la computación. Muchas de estas investigaciones se han encauzado a diferentes algoritmos de búsqueda, entre los que destacan los arboles binarios, los arboles B+ y la dispersión. Anteriormente se mencionó que la dispersión engloba varios métodos de búsqueda, destacando entre ellos la dispersión dinámica como uno de los más eficientes, pero su a vez, es uno de los que más han sufrido modificaciones.

Una de esas variantes se denomina dispersión dinámica virtual (DDV). Su principal característica es el manejo de la memoria virtual, usada con el objetivo de facilitar la inserción, la búsqueda y la eliminación de información. Logrando además realizar un mayor número de expansiones y contracciones del que se logra usando solamente la memoria física. La DDV centra su análisis sobre tres puntos principales : la función de dispersión, el manejo de la información y la memoria virtual.

4.1 Función de dispersión

Uno de los aspectos más trascendentales de la dispersión es la función que da su nombre a la estructura, no importando los elementos utilizados y la modalidad que se este desarrollando.

Para comenzar con nuestro análisis, debemos tener en cuenta la siguiente consideración [1] : existe un conjunto W de m palabras, que a su vez, es un conjunto finito de símbolos sobre un alfabeto ordenado Σ. Entonces se dice que una función de dispersión mapea un conjunto de palabras W en un intervalo de valores enteros I, lo se expresa de la siguiente forma:

$$h \ : \ W \rightarrow I \qquad \textit{Ecuación 4.1}$$

en donde I es un conjunto de valores enteros, que va desde 0 hasta $k-1$; siendo k un valor entero. De tal forma, la ecuación 4.1 se transforma en :

$$h \ : \ W \rightarrow [0, k-1] \qquad \textit{Ecuación 4.1.1}$$

pero en el caso de que $k = m$, se dice entonces que tenemos una función de dispersión mínima perfecta.

Con una función de dispersión mínima perfecta se pueden realizar manejos más depurados de la información en las localidades de almacenamiento primario y secundario. Pero en la mayoria de las ocasiones no se puede contar con una función de tales características, de ahí que se obtenga un número significativo de colisiones, por ende, la función de dispersión es necesario que tome al menos un cierto *O(1)* tiempo en lo que respecta a la lectura de las llaves, las cuales se procesan a través de un número constante de divisiones y multiplicaciones. Además, hay que tener en cuenta que la función almacenará las llaves en una capacidad de

$$O(m + \log\log u)\ bits \qquad \textit{Ecuación 4.2}$$

de espacio; siendo u el tamaño del universo U, dentro del cual se encuentran las llaves de búsqueda que han sido seleccionadas de W.

Por lo mismo, han surgido diversos esquemas y metodologías para encontrar la función ideal de dispersión en los términos antes citados. Estos métodos ham sido clasificados de muy diversas maneras. Una de estas clasificaciones es la siguiente [1] :

- ✓ Segmentación
- ✓ Restricción de espacio
- ✓ Matrices de compactación

La función que se desarrolla para este caso tiene su fundamento en los esquemas basados en la restricción de espacio. Cichelli [2] es uno de los personajes que mejor ilustran dicho método, pues presenta un modelo heurístico para un conjunto de llaves predeterminado. Su función

$$h(w) = length(w) + g(w[1]) + g(w[length(e)]) \quad \textit{Ecuación 4.3}$$

opera bajo la manipulación de tres parámetros. El primero de ellos recibe la longitud de la cadena, mientras que el segundo parámetro recibe el valor significativo del primer caracter, y el tercero obtiene el valor significativo del último caracter. En la función se observa que la función g, que es un parámetro de la propia función de dispersión.

Y como se habrá podido notar, el método es muy simple. Pero desafortunadamente, tiene una

seria limitante en relación con el tiempo exponencial utilizado para el paso de búsqueda de las llaves. El algoritmo trabaja bien cuando el conjunto de palabras es muy pequeño. Un estudio Monte Carlo demostró que él método de Cichelli para una $k > m$, la probabilidad de éxito del método tendía a cero.

A este método se le han hecho diversas mejoras e inclusive se han propuesto diversos métodos alternos alantes citado, esto es con el fin de poder obtener un conjunto de palabras mayor.

Los anteriores criterios han servido como soporte para el desarrollo de diversos esquemas que emplean funciones de dispersión. Como es el caso de Michael J. Folk y Bill Zoellick [3], los cuales proponen un método bastante interesante en la forma de trabajar la llave de entrada (ver esquema 4.1).

```
Suma ← 0;
Si Longitud(Cadena) < 16 entonces
Para Longitud(Cadena) hasta 16
    Concatena(Cadena, ' ');
Para x ← 0 Hasta Longitud(Cadena)
    Suma ← (Cadena[x]×100+Cadena[x+1]) mod 19937;
    x ← x+2;
Convertir Suma en Binario;
```

Esquema 4.1 Proceso de transformación

Este algoritmo se basa en criterios ya antes mencionados, los cuales se ven reflejados de manera directa en su algoritmo. Para empezar, ellos consideran el manejo de números enteros como un punto esencial de procesar las llaves, pues el manejo de registros a través de la función de dispersión no puede exceder de 65,536, es decir, un número de 16 bits.

Este proceso se puede ver en la suma de los valores ASCII (por sus siglas en ingles, *American Standard Code for Information Interchange*) de cada una de las letras que componen a la cadena que funge como llave de entrada. Además, al ir realizando la suma de dichos valores, se aplica un proceso de aritméica modular cada vez que avanza el contador. De tal manera el proceso de transformación recae en la conversión de una cadena que ocupa 128 bits de espacio por una variable que ocupa 16 bits. Un segundo proceso de dispersión es realizado, pues la función presenta un número de colisiones bastante alto. Y cuyo dilema se resuelve por medio

de una serie de corrimientos en el valor obtenido. Logrando con esto determinar la dirección final que ha de usarse en el almacenamiento o recuperación de la información.

4.1.1 Diseño

El diseño de la función de dispersión retoma los aspectos más relevantes de los criterios de espacio y tiempo antes mencionados, los cuales definen tanto el almacenamiento de las llaves en memoria como en disco duro, y su vez, el desempeño que va a tener la función al momento de trabajar. Por consiguiente, la función de dispersión de sdefine bajo los siguientes puntos :

- ✓ Un codigo de símbolos.
- ✓ Una longitud máxima de la llave de entrada.
- ✓ Limites de almcenamiento para las llaves.

Una vez establecidas las propiedades que ha de tener la funciónde dispersión, es conveniente explicar el porque de las mismas. Para el caso del prime punto, se han desarrollando e implementado varias funciones de dispersión que emplean tablas de códigos, siendo la más utilizada la tabla de códigos ASCII.

De esta manera, todos los procesos de conversión que emplean este tipo de tablas procuran obtener un valor entero a través de una serie de operaciones aritméticas como matemáticas predefinidas. Donde el número de operaciones aritméticas es bastante alto en comparación con el número de operaciones matemáticas.

Logrando con esto reducir los tiempos de procesamiento y los espacios de almacenamiento de las llaves en memoria. Pero la distribución de los valores obtenidos no es lo bastante uniforme, es decir, que el número de colisiones es bastante siginificativo.

De ahí se desprende la razón de porque utilizar una nueva tabla de codigos, la cual es una excepción de las anteriores, en vez de asignar un valor entero, asume un valor de tipo real. Ya que al manejar los valores de tipo real en operaciones aritméticas así como en matemáticas, el

número que se obtiene de las mismas operaciones ofrece la flexibilidad de obtener un número entero, cuyo porcentaje de aleatoriedad es mayor, siendo el caso contrario con el manejo de números enteros.

Al usar este tipo de valores, se sacrifican tiempos de ejecución y espacios de procesamiento. Pero se obtiene una mejor distribución de los valores recabados a partir de la función de dispersión.

Dentro del segundo punto se define la longitud máxima que debe tener la llave de entrada. Por conveniencia y sentido lógico, la longitud de la llave no debe ser mayor a 16 caracteres. Por último, la aplicación requiere de poder utilizar valores binarios. Como algunos lenguajes de programación no involucran una transición directa de un valor decimal a uno binario, es necesario crear este proceso. Así, el valor que rertome este proceso será a través de una cadena de caracteres que presente de esta manera simbólica el número binario. Pero como se considera la limitante de espacio, la cadena solo ocupará el espacio necesario para almacenar la conversión del número.

Una vez realizadas las explicaciones pertinentes, proseguimos con nuestro diseño, donde la propuesta del nuevo código la definimos en base a tres grandes conjuntos y un elemento adicional, los cuales conforman el universo de elementos que podrá mapear la función de dispersión :

$$U = \{S\_y\_N \cup L\_M \cup L\_m \cup E\_b\} \qquad \textit{Ecuación 4.4}$$

siendo :

S_y_N = { !, #, $, ... , 0, 1, ... , 9 };
L_M = { A, B, C, ... , Z} ;
L_m = { a, b, c, ... , Z } ;
E_b = Espacio en blanco ;

Donde *S_y_N* es el conjunto de símbolos y números, *L_M* representa al conjuno de las letras mayúsculas, *L_m* es el conjunto de las letras minúsculas y *E_b* es el espacio en blanco.

El universo propuesto consta de 78 elementos, los cuales definen en su gran mayoría las palabras o cadenas que han de utilizarse como llaves de entrada. Además, comó se mencionó en parrafos anteriores, los símbolos se mapearán por medio de valores reales, que oscilan entre el 0,1 hasta el 9,9. (ver el apéndice A para mayor referencia).

La razón de utiliza de este rango rango y no uno más amplio, se puede apreciar de manera más clara en los procesos matmáticos que han de ser llevados a cabo. Estos se dividen en 2 partes : la primera parte trabaja con potencias, mientras que la segunda parte emplea funciones logarítmicas.

La primera fase de transformación involucra el código establecido para los caracteres. Los cuales serán procesados mediante las siguientes ecuaciones :

$$Suma \leftarrow Suma + \sqrt{e^{Caracter_x}} \qquad \textit{Ecuación 4.5}$$

$$Suma \leftarrow Suma + \sqrt{\pi^{Carcater_y}} \qquad \textit{Ecuación 4.6}$$

siendo :

$x \leftarrow k;$ *Los caracteres pares.*
$y \leftarrow k+1;$ *Los caracteres impares.*

mientras que :

$k \leftarrow k+2;$ *Representa la posición del caracter en la cadena.*

Esta distribución se da más que nada por las constantes ya empleadas. Estas constantes poseen una naturaleza única, donde el número de decimales que se emplean no representan una periodicidad y no existe un límite total al número de decimales a emplear. Así, al elevar un número con tales características a una potencia, y despues obtener su raíz cuadrada, el resultado conserva las propiedades de las constantes.

De tal forma, el primer proceso matemático se realiza por medio de las ecuaciones 4.5 y 4.6 hasta alcanzar el fín de cadena. Una vez que se ha alcanzado el fín de cadena, el segundo pro-

ceso matemático empieza a trabajar. El cuál consiste en un producto de logaritmos :

$$Valor_D \leftarrow \ln(Suma) \times \log(\pi^{Longitud}) \qquad Ecuación\ 4.7$$

El primer factor emplea la suma obtenida del primer proceso, a la cual se le aplica la función del *logaritmo* natural o neperiano. Mientras que al segundo factor , que es un proceso compuesto. Primero se eleva a π a la longitud de la cadena, y una vez que se cuenta con el resultado, se le aplica la función del *logaritmo común*. El proceso en sus dos fases termina al realizar el producto de ambos logaritmos.

Una vez concluida las fases matemáticas, se procede a la obtención del *número entero*; cuyo factor depende en parte del número de decimales que se deseen emplear para obtener el valor deseado. Supongamos que se tiene el número : 23.012496214564, se elimina la parte entera, quedando ahora en : 0.012496214564. Si desean manejar 3 decimales de discriminación, el valor entero será el 496. Por último, procede la conversión del número entero en base decimal a su correspondiente en base binaria.

Hasta *aquí* se han cubierto dos de las tres propiedades que definen a la función de dispersión. Para el tercer punto, la tabla 4.1 muestra la longitud que ha de tener la cadena simbólica que representará al número binario.

Aquí surge entonces la pregunta sobre la restricción de espacio : ¿ Él *porqué* se menciona y no se toma en cuenta ?, Porque si deseáramos manejar los valores tal como los arroja la función en su primer módulo, es decir, tomar el valor entero, es valilda la restricción, pero como se necesita manejar valores binarios, y en base a un anterior consideración de los lenguajes de programación, se recurre a un artificio para lograr este fin. Pero aún así se toma en cuenta la restricción de espacio. Al observar la tabla 4.1, notará que las longitudes son variables y no de un tamaño fijo. Pues creemos conveniente manejar las cadenas de esta manera para ahorar espacio en memoria como en disco duro. Aunque no es la más deseable pero si el más aceptable para esta situación.

Tabla 4.1 Longitud de la cadena en base a los valores de dispersión

De	Hasta	Longitud
0	1	1
2	3	2
4	7	3
8	15	4
16	31	5
32	63	6
64	127	7
128	255	8
256	511	9
512	1023	10
1024	2047	11
2048	4095	12
4096	8191	13
8192	16383	14
16384	32767	15
32768	65535	16

4.2 Memoria virtual

En esta sección se describirá una de las técnicas más usuales en el terreno de la administración de la memoria : la memoria virtual. La cuál se encuentra implementada en los moderno sistemas operativos (SO), como es el caso de Windows 95/NT, Unix, Mac OS, Open VMS, etcétera. Mientras que otros SO no fuerón dessarrollados para llevar a cabo esta tarea, siendo MS-DOS el clásico ejemplo.

Para poder describir mejor su funcionamiento y posterior inclusión dentro de la DDV, se delimita la técnica en los siguientes puntos : antecedentes, desempeño e integración. Cada uno de estos puntos ofrece una visión muy global de la técnica, para así poder comprender el uso que tendrá posteriomente dentro de la DDV.

4.2.1 Antecedentes

La necesidad cada vez más imperiosa de ejecutar programas grandes y el crecimiento en poder de las unidades centrales de procesamiento, empujaron a los diseñadores de SO's a desa-

rrollar e implementar una herramienta capaz de ejecutar automáticamente programas más grandes que la memoria real disponible, esto es, de ofrecer "memoria virtual" [4].

La memoria virtual se llama así porque el usuario final ve una cantidad de memoria mucho mayor que la real, y en realidad, se trata de la suma de la memoria de almacenamiento primario y una cantidad determinada de almcenamiento secundario [4]. El SO, en su módulo de manejo de memoria, se encarga de intercambiar programas enteros, segmentos o páginas entre la memoria real y el medio de almacenamiento secundario.

La memoria virtual se apoya en varias técnicas para lograr su objetivo. Una de las teorías más fuertes es la del "conjunto de trabajo", la cual se refiere aun programa o proceso que no este usando todo su espacio de direcciones en todo momento, sino que existen un cojunto de localidades activas que conforman el "conjunto de trabajo". Si se logra que las páginas o segmentos que contiene al cojunto de trabajo esten siempre en RAM, entonces el programa se desempeñara muy bien.

Otro factor importante es si los programas muestran un fenómeno llamdo "localidad", lo cual quiere decir que algunos programas tienden a usar mucho las instrucciones que estan cercanas a la localidad de la instrucción que se está ejecutando actualmente.

4.2.2 Desempeño

El desempeño de la memoria virtual depende básicamente de dos métodos : la paginación y la segmentación. Cada una de estas se encuentran implementadas por separado o en conjunto dentro delos SO's. Así mismo, se describen los problemas a los cuales se enfrenta el desempeño de la memoria virtual, como lo es la relocalización y la protección de las páginas.

4.2.2.1 Paginación

La paginación se hace en la memoria y consiste en que SO divide dinámicamente los programas en unidades de tamaño fijo, siendo generalmente múltiplos de 1 kilobyte, los cuales va a manipular desde la memoria al disco duro y viceversa. La proceso de intercambiar páginas,

segmentos o programas completos entre la memoria y el disco duro se le conoce como "*intercambio*".

En la paginación, se debe cuidar el tamaño de las páginas , ya que si estas son muy pequeñas, el control por parte del SO para saber cuáles están en RAM y cuales en disco, sus direcciones reales, etcétera; crece y provoca una "sobrecarga". Por otro lado, si las páginas son muy grandes, la sobrecarga disminuye, pero entonces puede ocurrir que se desperdicie memoria en pequeños procesos.

Uno de los aspectos más importantes de la paginación, así como de cualquier esquema de memoria virtual, es la forma de traducir una dirección virtual a una dirección real. Al tener una dirección *v=(b,d)*, la cuál está formada por un número de página virtual "b" y de un desplazamiento "d". Por ejemplo, si el sistema ofrece un espacio de direcciones virtuales de 64 kilobytes, con páginas de 4 kilobytes y la RAM sólo es de 32 kilobytes, entonces tenemos 16 páginas virtuales y 8 reales. La tabal de direcciones virtuales contiene 16 entradas, una por cada pagína virtual. En cada entrada o registro de la tabla de direcciones virtuales se almacenan varios datos : si la página está en disco o en memoria, que programa es el dueño de la página, si la página ha sido modificada ó es de lectura nada más, etcétera. Pero el dato que nos interesa ahora es el número de página real que le corresponde a la página virtual. Obviamente, de las 16 virtuales, soló 8 tendrán un valor de control que dice que la página está cargada en RAM, así como la dirección real de la página. Por ejemplo, supóngase que para la página virtual número 14 la tabla dice que efectivamente está cargada y es la página real 2 (dirección de memoria 8192). Una vez encontrada la página real, se le suma el desplazamiento, que es la dirección que deseamos dentro de la página buscada *(b'+d)*.

Cuando se está buscando una página cualquiera y está no est esta cargada, surge lo que se llama un "*fallo de página*". Este es caro para el manejador de memoria, ya que tiene que realizar una serie de pasos extra para poder resolver la dirección deseada y darle su contenido a quien lo pide. Los pasos que ha de cumplirse son los siguientes :

- ✓ Se detecta que la página no está presente y entonces se busca en la tabla de dirección de esta página en disco. Una vez localizada en disco se intenta cargar

en alguna página libre de RAM.

- ✓ Si no hay páginas libres se tiene que escoger alguna para enviarla hacia el disco. Una vez escogida y enviada a disco, se marca su valor de control en la tabla de direcciones virtuales para indicar que ya no esta en RAM, mientras que la página deseada se carga en RAM y se arca su valor para indicar que ahora ya está en RAM.

Estos procesos en conjunto son caros, ya que se sabe que los accesos a disco duro son del orden de decenas más lentos que a RAM.

En el ejemplo anterior se mencionó que cuando se necesita descargar una página de RAM hacia disco se debe de hacer una elección. Para realizar está elección existen varios algoritmos, los cuales se describen enseguida [4] :

- ✓ **La primera en entrar, primera en salir**. Se escoge la página que haya entrado primero y esté cargada en RAM. Se necesita que en los valores de control se guarde un dato de tiempo. No es eficiente porque no aprovecha ninguna característica de ningún sistema. Es justa e imparcial.

- ✓ **La no usada recientemente**. Se escoge la página que no haya sido usada en el ciclo anterior. Pretende aprovechar el hecho de la localidad en el conjunto de trabajo.

- ✓ **La usada menos recientemente**. Es parecida a la anterior, pero escoge la página que se ha usado hace más tiempo; pretendiendo que como ya tiene mucho tiempo sin usarse es muy probable que siga sin usarse en los próximos ciclos. Necesita de una búsqueda exhaustiva.

- ✓ **La no usada frecuentemente**. Este algoritmo toma en cuenta no tanto el tiempo, sino el número de referencias. En esta caso cualquier página que se use muy poco, menos veces que alguna otra.

- ✓ **La menos frecuentemente usada**. Es parecida a la anterior, pero aquí se busca en forma exhaustiva aquella página que se ha usado menos que todas las

demás.

- ✓ **En forma aleatoria.** Elige cualquier página sin aprovechar nada. Es injusta, imparcial e ineficiente.

Otro dato interesante de la paginación es que ya no se requiere que los programas estén ubicados en zonas de la memoria adyacente, ya que las páginas pueden estar ubicadas en cualquier lugar de la memoria principal.

4.2.2.2 Segmentación

La segmentación se aprovecha del hecho de que los programas se dividen en partes lógicas, como son las partes de datos, de código y de la pila. La segmentación asigna particiones de memoria a cada segmento de un programa y busca como objetivos el hacer fácil de compartir segmentos (por ejemplo, las librerías compartidas) y el intercambio entre la memoria y los medios de almacenamiento secundario.

Por ejemplo, en la versión de Unix SunOS 3.5 no existían librerías compartidas para algunas herramientas, por citar, para los editores de texto orientados al ratón y menús. Cada vez que un usuario invocaba a un editor, se tenía que reservar un megabyte de memoria. Como los editores son una herramienta muy solicitada, se dividió en segmentos para la versión 4.x (que a su vez se divide en páginas), pero lo importante es que la mayor parte del editor es común para todos los usuarios. De manera que la primera vez que cualquier usuario lo invocaba, se reservaba 1 megabyte de memoria como antes, pero para el segundo, tercero y el resto de los usuarios que invocaban al editor, sólo comsumían 20 kilobytes extra de memoria. El ahorro es impresionante.

Obsérvese que en la segmentación las particiones de memoria son de un tamaño variable, en contraste con las páginas de tamaño fijo en la paginación. También se puede decir que la segmentación tiene una granularidad menor que la paginación por el tamaño de segmento versus el tamaño de páginas.

La traducción de direcciones virtuales a direcciones reales es igual que la llevada a cabo por la

paginación, tomando en consideración que el tamaño de los bloques a controlar por la tabla de traudcción son variables, por lo cual, cada entrada en dicha tabla debe contener la longitud de cada segmento a controlar. Otra vez se cuenta con un registro base que contiene la dirección del comienzo de la tabla de segmentos.

La dirección virtual se compone de un número de segmento *s* y un desplazamiento *d* para ubicar un byte o palabra dentro de dicho segmento. Es importante que el el desplazamiento no sea mayor que el tamaño del segmento, lo cual se controla simplemente checando que ese valor sea mayor que la dirección del inicio del segmento que el inicio sumado al tamaño.

Una vez dada una dirección virtual $v=(s,d)$ se realiza la operación $d+s$ para hallar el registro (o entrada de la tabla de segmentos) que contiene la dirección de inicio del segmento de la memoria real, denotada por s'. Ya conociendo la dirección de inicio en memoria real s' solo resta encontrar el byte o palabra deseada, lo cual se hace sumandole a s' el valor del desplazamiento, de modo que la dirección real "r" sea $r=s'+d$. Así mismo, se logra establecer los campos que indican la longitud, los permisos, la presencia o ausencia y dirección de inicio en memoria real del segmento.

Según amplios experimentos, sugieren que un tamaño de páginas de 1024 bytes generalmente ofrece un desempeño muy aceptable. Intuitivamente parecería que el tener páginas de mayor tamaño harían que el desempeño fuera óptimo, pero no es así, a menos que la página fuera del tamaño del proceso total. No es así con tamaños grandes de páginas menores que el proceso, ya que cuando se trae a memoria principal una página por motivo de un byte o palabra, se estan trayendo muchísimos bytes de los deseados.

Un hecho notable en los sistemas que manejan paginación es que cuando el proceso comienza a ejecutarse, ocurre un gran número de fallos de página. Porque es cuando está referenciando muchas direcciones nuevas por primera vez, después el sistema se estabiliza, conforme el número de marcos se acerca al tamaño del conjunto de trabajo.

4.2.2.3 El problema de la relocalización

Este problema consiste en que los programas que se necesitan cargarse a memoria real ya están compilados y ligados, de manera que internamente contienen una serie de referencias a direcciones de instrucciones, rutinas y procedimientos que ya no son validas en el espacio de direcciones de memoria real de la sección en la que carga el programa. Esto es, cuando se compiló el programa se definierón o resolvierón las direcciones de memoria de acuerdo a la sección de ese momento, pero si el programa se carga otro día en una sección diferente, las direcciones reales ya no coinciden.

En esta caso, el manejador de memoria puede solucionar el problema de dos maneras :

- ✓ La solución **estática** consiste en que todas las direcciones del programa se vuelvan a recalcular al momento en que el programa se carga en memoria, esto es, prácticamente se vuelve a recompilar el programa.

- ✓ La solución **dinámica** consiste en tener un registro que guarde la dirección base de la sección que va a contener al programa. Cada vez que el programa haga una referencia a una dirección de memoria, se le suma el registro base para encontrar la dirección real. Por ejemplo, suponga que el programa es cargado en una sección que comienza en la dirección 100. El programa hará referencias a las direcciones 50, 52 y 54. Pero el contenido de esa direcciones no es el deseado, sino las direcciones 150, 152, y 154, ya que ahí comienza el programa. La suma 100+560, ... , Etcétera, se hacen al tiempo de ejecución.

La primera solución vale más la pena que la segunda si el programa contiene ciclos y es largo, ya que consumirá menos tiempo en la resolución inicial que la segunda solución en las resoluciones en linea.

4.2.2.4 El problema de la protección

Este problema se refiere a que, una vez que un programa ha sido cargado a memoria en algún segmento en particular, nada le impide al programador que intente direccionar (por error o deliberadamente) localidades de memoria menores que el límite inferior de su programa o supe-

riores a la dirección mayor, es decir, quiere referenciar localidades fuera de su espacio de direcciones. Obviamente, este es un problema de protección, ya que no es legal leer o escribir en áreas de otros programas.

La solución a este problema también puede ser el uso de un registro base y un registro límite. El registro base contiene la dirección donde termina. Cada vez que el programa hace una referncia a memoria se checa si cae en el rango de los registros y si no es así, se envúia un mensaje de error y se aborta el programa.

4.2.2.5 Sistemas combinados

La paginación y la segmentación son métodos de manejo de memoria bastante efectivos, aunque la mayoría de lso SO's modernos implantan esquemas combinados, es decir, combinan la paginación y la segmentación. La idea de combinar estos esuqemas se debe a que de esta forma se aprovechan los conceptos de la división lógica de los programas (segmentos) con la granularidad de las páginas. De está forma, un proceso estará repartido en la memoria real en pequeñas unidades (páginas) cuya liga son los segmentos. También es factible así el compartir segmentos a medida que las partes necesitadas de los mismos se van referenciando.

Para comprender este esquema, nuevamente se verá cómo se traduce una dirección virtual a una localidad de memoria real. Para la paginación y la segmentación se puede decir que el direccionamiento es "bidimensional", porque se necesitan dos valores para hallar la dirección real. Para el caso combinado, se puede decir que se tiene un direccionamiento "tridimensional".

El sistema debe contar con una tabla de procesos (TP). Por cada región de esa tabla se tiene un número de proceso y una dirección a una tabla de segmentos. Cuando un proceso hace alguna referencia a memoria, se consulta la TP para encontrar la tabla de segmentos de ese proceso. En cada tabla de segmentos de proceso (TSP) se tienen los numeros de los segmentos que componen a ese proceso. Por cada segmento se tiene una dirección a una tabla de páginas. Cada tabla de páginas tiene las direcciones de las páginas que componen a un sólo segmento. Por ejemplo, el segmento *A* puede estar formado por las páginas reales '*a*', '*b*', '*c*', '*p*' y '*x*'.

El segmento *B* puede estar compuesto de las páginas *'f'*, *'g'*, *'j'*, *'w'* y *'z'*.

Para traducir una dirección virtual *v=(s,p,d)* donde "s" es el segmento, "p" es la página y "d" es el desplazamiento en la página; donde se hace lo siguiente : primero, se ubica de qué proceso es el segmento y se localiza en la tabla de segmentos de se proceso en la TP. Con "s" como índice se encuentra un renglón (registro) en la tabla de segmentos de ese proceso y en ese renglón está la dirección de la página en memoria real. Una vez en esa dirección de memoria real se encuentra el byte (ó palabra) requerido por medio de valor de "d".

Ahora, en este esquema pueden haber dos tipos de fallos : fallo por página y fallo por segmento. Cuando se hace referencia a una dirección y el segmento que la contiene no esta en RAM (aunque sea parcialmente), se provoca un fallo por falta de segmento y lo que se hace es traerlo del medio de almacenamiento secundario y crearle una tabla de páginas. Una vez cargado el segmento se necesita localizar la página correspondiente, peró esta no existe en RAM, por lo cual se provoca un fallo por página y se carga de disco, y finalmente se puede ya traer la dirección deseada por medio del desplazamiento de la dirección virtual.

La eficiencia de la traducción de direcciones tanto en paginación, segmentación y esquemas combinados se mejora usando memorias asociativas para las tablas de páginas y segmentos, así como memorias cache para guardar los mapeos más solicitados.

Otro aspecto importante es la estrategia para cargar páginas o segmentos a la memoria RAM. Se usan dos estrategias para tal fin : el cargado de páginas por demanda y elcargado de páginas anticipada. La estrategia de cargado por demanda consiste en que las páginas solamente son llevadas a RAM si fueron solicitadas, es decir, si se hizo referencia a una dirección que cae dentro de ellas. La carga anticipada consiste en tratar de adivinar qué páginas serán solicitadas en el futuro inmediato y cargarlas de antemano, para cuando se pidan ya no ocurran fallos de página. Ese "adivinar" puede ser que se aproveche el fenómeno de localidad y que las páginas que se cargan por anticipado sean aquellas que contienen direcciones contiguas a la dirección que se acaba de referenciar.

4.2.3 Integración

Antes de empezar este punto, es conveniente dar antes algunas definiciones :

- ✓ **Costo por acierto : H** . Es el tiempo que se toma en accesar la memoria virtual en el caso de que no este presente un fallo de página. Esto se hace bajo la suposición de que el costo de translación es igual a cero, entonces H es igual al tiempo de acceso a la memoria física

- ✓ **Relación de acierto : R** . Es la relación entre el número de veces en que la memoria virtual es accesada sin dar origen a un fallo de página. Siendo este un concepto similar a la relación de acierto, el cual es el concepto del promedio entre el tiempo y los fallos, que es igual a $H \times R$, si no consideramos el tiempo que requiere tomar el fallo.

- ✓ **Castigo por fallo : M** . Tiempo que toma en accesar la memoria virtual en el caso de tener un fallo de página.

- ✓ **Relación de velocidad : S** . Esta relación es entre el castigo por fallo y el costo por acierto. Por ejemplo, si tenemos un costo por acierto de 100 nanosegundos y un castigo por falla de10 milisegundos, entonces la relación de velocidad es de 100,000.

De esta manera es fácil de observar que el número de accesos espectante a la memoria virtual se describe a través de la siguiente ecuación :

$$E(T_A) = \frac{R \times H + M}{R+1} \sim= \frac{R \times H + M}{R} = H + \frac{M}{R} = H\left[1 + \frac{S}{R}\right]$$

Ecuación 4.8

Por ejemplo, si H es de 0.1 microsegundos, M es de 15 milisegundos y R es de 100,000, entonces el tiempo de acceso esperado (T_A)a la memoria virtual es de 0.25 milisegundos, esto es, 2.5 veces que se accesa la memoria.

En el manejo de la memoria virtual no sólo participa la memoria principal y el almacenamiento secundario, sino que tambien la memoria cache es participante. Siendo parte relevante dentro del esquema básico de la administración global de la memoria. En todos los casos es bueno el trabajo realizado, porque muestran un aspecto de "localidad" en su comportamiento (Note que cuando se habla de localidad, se hace referencia de una localidad temporal o de una localidad de memoria. La localidad temporal hace referencia si se ha usado recientemente el archivo de intercambio; mientras que la localidad de memoria es con referencia al uso que se tenga en ese instante el archivo de intercambio - *ver tambien el apartado dedicado a los Sistemas combinados*-).

Se permite considerar la combinación del cache y la memoria principal en la translación de dlas direcciones virtuales. Teniendo Hc, Rc y Mc por un lado; mientras que por otro lado tenemos Hm, Rm y Mm. Pueden ser respectivamente el costo de acierto, la relación de acierto y el castigo por fallo en el caso del cache a la memoria y de la memoria principal al direccionamiento virtual en disco.

Ante este punto, es bueno considerar dos casos : el caso en que el espacio de direccionamiento virtual en cache opera en la parte alta de la memoria principal en el espacio de direccionamiento virtual. El otro caso es el inverso.

En el caso de que el cache este operando en el direccionamiento virtual (por ejemplo, el mantener pares de la forma [Direcciones virtuales, valor]), el tiempo de acceso esperado en el sistema de memoria resultante desde el cache al disco es :

$$E(T_A) \sim= Hc + \frac{Mc}{Rc} = Hc + \frac{I}{Rc}\left[Hm + \frac{Mm}{Rm}\right]$$

$$\sim= Hc + \frac{Hm}{Rc} + \frac{Mm}{Rc + Hm} \qquad \textit{Ecuación 4.9}$$

Por ejemplo, si usamos los valores antes mencionados para Hm, Rm, Mm y asumimos que el Hc es de 0.02 microsegundos y rc es de 18, entonces tenemos que tomar el tiempo de acceso esperado es de alrededor de 0.03 microsegundos.

Mientras que en el caso inverso (por ejemplo, mantener pares de la forma [Direccionamiento físico, valor]), el tiempo de acceso en el sistema de memoria resultante desde el cache hacia el disco es del orden de :

$$E(T_A) = E(T_{A'}) + \frac{Mm}{Rm} = Hc + \frac{Hm}{Rc} + \frac{Mm}{Rm} \qquad \textit{Ecuación 4.10}$$

Para los mismo valores antes citados, ahora los tiempos de acceso es del orden de $0.026 + 0.15$ microsegundos. Cómo se puede apreciar en los ejemplos, el primer caso resulta ser más eficiente que el segundo.

Para que estas fórmulas tengan efecto, se debe tener en cuenta el tipo de fragmentación que se haga del programa. Existen dos tipos de fragmentación :

- ✓ **Interna**. Suponga que usted está trabajando la memoria virtual por páginas, de un tamaño p. Entonces, si se tiene un programa de un tamaño s, se requieren $Ceiling(s/p)$ páginas. De esta forma, el desperdicio se define de la siguiente forma :

$$p \times Ceiling(s/p) - S \ \ bytes \qquad \textit{Ecuación 4.11}$$

- ✓ **Externa**. Ahora suponga que usted trabaja la memoria virtual por segmentación. Entonces el almacenamiento es destinado por bloques de tamaños variables. De esto resulta la creación de bloques de tamaños pequeños, que resultan ser hoyos dentro del almacenamiento. Estos hoyos representan un desperdicio en el almacenamiento. Inclusive existen diversas políticas para evitar estos desperdicios o en su defecto, tener el mínimo desperdicio en el almacenamiento.

En este punto tambien es conveniente tener en cuenta el hardware que de soporte a este tipo de manejos para la memoria virtual. Ahora, en los modernos equipos se cuenta con una unidad administradora de la memoriaque realiza la traslación de las direcciones virtuales a las físicas. Esta lista ejemplifica otras labores que realizan estas unidades :

- ✓ Interrupción en el caso de que se presente un falló de página con información para identificar que fue lo que lo originó.
- ✓ Soporte para las políticas de reemplazo.
- ✓ Soporte para seleccionamiento cuando se tenga activa la memoria virtual.
- ✓ Soporte para la eliminación de errores cuando se tenga activa la memoria virtual (seleccionando las direcciones que han de ser trasladadas y cuales no).
- ✓ La translación de las direcciones en diversos espacios de direccioanmiento virtual.

4.3 Diseño del árbol

El esquema empleado para la construcción de arboles dinámicos, utilizando búsqueda multinivel, ha sido aplicado con la finalidad de resolver problemas de generación de códigos existentes dentro de algunos lenguajes de programación, para algunos casos de declaraciones especificas. Este método ha tenido ventajas, sobre algunos métodos preexistentes, requiriendo menos de tres accesos en promedio para la búsqueda por rama.

Este método puede ser readecuado, para llevar a cabo la generación de arboles radix de búsqueda multinivel, este tipo de arboles pueden sustituir de manera apropiada algoritmos de búsqueda, en donde se utilizan arboles estáticos. De igual forma se brinda un esquema de búsqueda similar, pero orientado hacia el uso de los arboles radix binarios.

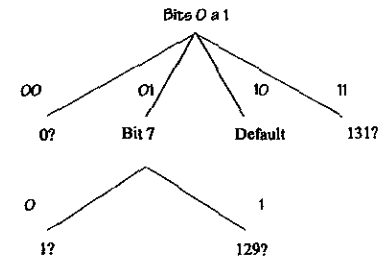
Consideremos el siguiente problema : la necesidad de implementar una búsqueda de cualquiera de las m subcadenas de longitud L, que componen a una cadena S, por un momento asumamos la idea, de que cada uno de los valores de dispersión se encuentra calculado para cada una de las m subcadenas; esto a través de la implementación de una función de dispersión especial. Con fundamento en lo anterior, la búsqueda puede ser concluida mediante una sencilla detección de cada uno de los miembros componentes de S, es decir, a través del calculo de los valores para cada conjunto de letras de L, los miembros adyacentes dentro de S, se terminan

procesando de manera semejante al algoritmo de Karp-Rabin. Posterior a la obtención de cada valor de dispersión, para cada elemento L adyacente de S, se realizara la comparación de cada uno de estos, contra los m valores de dispersión precalculados.

Nuestro objetivo es determinar la secuencia de bits critica, para la entrada de cadenas y que estas sean dispersadas sobre del árbol, esto realizado de manera recursiva a través de los subconjuntos de cadenas no triviales, que han sido detectados a través del proceso de dispersión.

La Figura 4.1 muestra un claro ejemplo de esto. Los casos de los números 0, 1, 129 y 131 son determinados casi únicamente por la diferencia dentro de los dos últimos bits, la única excepción es el par del 1 y 129; en donde el séptimo bit ayuda en el determinar de la diferencia. Con base en lo anterior podemos evitar que los candidatos de entrada sean dispersados primeramente en sus dos bits menos significativos, y entonces, si es necesario dentro del séptimo bit, que es considerado el bit más significativo. Finalmente podremos realizar una comparación, para poder verificar si el valor de entrada es un residuo del dato de entrada que va a ser dispersado. Hay que hacer notar que el código opta por dirigirse directamente hacia el manejador por default, por lo que para cualquier tiempo de ejecución de elementos en la entrada se finalizara en un tiempo equivalente a 10_2, lo que hace considerar que ningún caso caerá dentro del determinado patrón de bits.

Es deseable utilizar páginas de memoria lo suficientemente amplias, en determinado orden, con el objetivo de hacer que el árbol de tenga la posibilidad de expanderse o contraerse, ya que debemos de limitar el tamaño de las páginas implementadas, ya que las ramas, requieren del uso de tablas de dispersión que crezcan exponencialmente dentro de todo el tamaño asignado al manejo de una página. Para tal caso se recomienda la utilización de un algoritmo que permita determinar el tama-



Caso	Binario
0	0 00000 00
1	0 00000 01
129	1 00000 10

Fig. 4.1 Esquema representativo del árbol radix.

ño de lo que hemos denominado "una página critica", dentro de la cual se almacenaran las incidencias que necesitan un mayor número de subconjuntos y de espacio dentro de la memoria; al menos, mas de la con que cuenta la página estándar.

4.3.1 Definiciones

Dado Z el conjunto de todos los K valores de bits, donde K es un entero positivo. Tenemos que C es subconjunto de Z, para formar un conjunto de incidencias, M un conjunto de etiquetas o marcadores y m_d una etiqueta adicional. Se asume como entrada un conjunto $P = \{(c_i, m_i)\}$ de pares ordenados, donde cada c_i pertenece al conjunto C, y se encuentra asociada a una sola etiqueta $m_i \in M$. Por lo que P se define como una función total de C hacia M.

Si deseamos generar el código para una función que desempeñe un mapeo de

$$F : Z \rightarrow M \cup \{m_d\}, \qquad \textit{Ecuación 4.12}$$

contemplando un valor de entrada $z \in Z$, y que va a ser mapeado dentro de m_d si $z \notin C$, por el contrario, si $z = c_i$, para $c_i \in C$, se tiene que dentro de la etiqueta $m_i \in M$, ha sido definida como elemento dentro del conjunto P. Implica que F desempeña la función de una palabra que entra.

Observe que los K valores de bits, dados por $b_{k-1}, \ldots, b_0$ y se tiene una página W, para almacenar una secuencia de bits $b_l, \ldots, b_r$, donde $K > l \geq r \geq 0$. Dado él par ordenado $val(s,W)$ que será conformado por el valor de los bits s, visibles dentro del espacio perteneciente a la pagina W. Lo que implica que, si $W = b_5, \ldots, b_3$, $val(41,W) = val(101001_2, W) = 101_2 = 5$.

Una página $W = b_l, \ldots, b_r$ es critica dentro de un subconjunto S de Z si la cardinalidad del conjunto $V_W = \{ val(s,W) \mid s \in S \}$ es mayor que 2^{l-r}. Entonces esto implica que W es critica, si su tabla de dispersión es de tamaño 2^{l-r+1}, y esta se encuentra ocupada mas arriba de la mitad de su capacidad. Una ventana W es mas critica si $|V_W| \geq |V_{W'}|$ para todas los longitudes de página W'.

4.3.2 Análisis de métodos

Ahora podemos comparar métodos a partir del análisis de los peores casos y esperar el tiempo que sea necesario para llevar a cabo su conclusión, asumiendo que sé esta ejecutando una incidencia con m palabras, y dicha incidencia es llamada n veces a través de entradas desde un archivo. Él numero de ramas es usado como el patrón de medición, para nuestras medidas de complejidad de tiempo [6].

- ✓ **Árbol binario**. Este método es esencialmente una búsqueda lineal. Por lo tanto toma $O(mn)$ tiempo, para la realización de las búsquedas.

- ✓ **Árbol binario balanceado**. El tiempo de ejecución para la entrada se compara con el tiempo de búsqueda de una incidencia, cuando el árbol de búsqueda es balanceado. Entonces el tiempo de ejecución es aproximadamente de $O(n \log m)$.

- ✓ **Árbol binario balanceado a tablas de dispersión**. Este es un árbol binario balanceado de búsqueda, para algunos casos la incidencia es extensa y se hace necesario contar con directorios de indexación o que el elemento sé divi-dida en rangos. Dado él numera de nodos y hojas con tablas de dispersión se tendría r. El peor tiempo de ejecución ocurriría cuando $r = m$ y es $O(n \log m)$, entonces $\log m$ es el tiempo de búsqueda para el árbol. El tiempo de ejecución total para la búsqueda es $O(n \log r)$.

- ✓ **Método de tabla de salto**. Esta es una tabla de dispersión directa y tiene un tiempo de ejecución de $O(n)$ en todos los casos. Esta requiere un espacio lineal para el manejo del rango de las incidencias de entrada, y por lo tanto es impractico, para el procesamiento de algunos conjuntos de entrada. Dada la base anterior este es usualmente el método mas optimo para el caso de conjuntos de elementos muy densos.

El análisis de este método es de alguna manera diferente, que el de los demás ya que el árbol, no es estrictamente balanceado y binario. En el peor de los casos, el conjunto de las incidencias de elementos, es un numero distinto a una potencia de dos. Entonces este árbol se esque-

matiza con una profundidad máxima de $K/2$, donde K es el tamaño de la arquitectura de la palabra. Por lo que entonces el peor tiempo de ejecución seria entonces de $O(nK)$. Este sigue siendo aun muy lineal en n.

Asumimos que los m casos de entradas se encuentran distribuidas de igual forma en K números de bits. Dada una página W de longitud L, la probabilidad de que sean r valores distintos, $val(c_i, W)$, donde c_i es en caso del conjunto de m ocurrencias de entrada y $0 < r < 2^L$, se encuentra dada por la siguiente ecuación [6] :

$$P(r,m,L) = \frac{2^L!}{(2^L - r)!\, 2^{mL}} \left\{ {m \atop r} \right\} \qquad \textit{Ecuación 4.13}$$

Entonces si nosotros tenemos m elementos con reemplazo, empleando una fuente aleatoria de valores dentro del rango $[0; 2^L)$, $P(r,m,L)$, lo anterior nos proporciona la probabilidad de que el ejemplo contenga r valores diferentes. Dado lo anterior es que se puede definir la probabilidad de que la pagina W es critica como [6]

$$P_{Critico}(m,L) = \sum_{r=2^{L-1}+1}^{2^L} P(m,r,L)$$

Cuando m llega a ser muy grande en tamaño $P_{critico}(m,L)$ se aproxima a la unidad, debido a $L=[\log m]$. Entonces esperamos que el mas alto rango de ramas de multinivel, tengan un factor fuera de orden m, con $m/2$ ramas conectadas a conjuntos de incidencias no triviales, y cada uno de estos contenidos dentro de dos incidencias. Por lo tanto el promedio de las ramas, que mantiene uniforme el conjunto de incidencias(tres), así como el brazo de alto nivel, además de un promedio de uno y medio ramas adicionales, para el manejo de cada subconjunto de elementos de entrada.

4.4 Referencias

[1] Gorge Havas and Bohdan S. Majewski. "Optimal algoritms for minimal perfect hashing". Technical report number 234. Departament of computer Science. University of Queensland. Australia. 1992.

[2] Cichelli, Richard J. Minimal Perfect Hash Functions made simple. Communications of the ACM. Vol 23; No. 12; December 1981. New York, New Jersey. Pags. 17-19.

[3] Bill Zoellick and Michale J. Folk. File Structures. *Addison - Wesley Reading*, Mass. 2nd. Ed.1992.

[4] Adminstración de la memoria. http://info.pue.udlap.mx/udla/clases/so/04.html. 1995.

[5] CIS 307 : Virtual Memory http://joda.cis.temple.edu/~ingargio/cis307/readings/virtrual.html.1997.

[6] Úlfar Erlingsson[a] , Mukkai Krishnamoorthy[a] and T. V. Raman[b]. "Eficient Multiway radix Search Tree". [a]Rensselaer Polytechnic institute, Troy, NY 12180. [b]Advanced Technology Group, Adobe Systems, Mountain View, CA 94039. Septiembre 1996.

Capítulo 5

Implementación

Partiendo de los principios establecidos dentro del pasado capítulo, relativo al análisis de la aplicación, se han realizado los trabajos encaminados a la implementación y desarrollo de las siguientes funciones, los cuales se consideran como fundamentales :

- ✓ **Función de dispersión.**
- ✓ **Árbol dinámico.**

Mientras que la administración de la memoria virtual va implícito dentro de la estructura del árbol. La descripción e implementación de estas funciones comprende la base algorítmica de la aplicación. Lo anterior se encuentra contenido dentro de un modulo, el cual adicionalmente comprende la utilización como base de operación de las políticas de almacenamiento y paginación para la administración de los recursos de memoria virtual, la cual ha sido implementada para su funcionamiento en Windows 3.x estándar, y que debido a las actualizaciones funciona dentro del nuevo esquema implementado en Windows 95.

Dentro de este capitulo se presenta una visión especifica acerca del diseño y construcción de estas funciones, a través de la descripción y seguimiento de cada uno de los procesos realizados, lo anterior con la finalidad de ejecutar la tarea deseada. En lo que se refiere a la función de dispersión, se analiza mas a profundidad su diseño y construcción, así como la base aritmético y lógica, utilizada para el diseño y conformación del algoritmo que realiza la distribución de los elementos dentro del árbol dinámico implementado dentro de la memoria.

De igual forma se observan los aspectos relativos a la implementación de la estructura de almacenamiento dentro de la memoria del equipo, en donde se establecen una serie de conceptos e ideas acerca de la concepción del árbol, del nodo y de la información que se encuentra contenida dentro de este ultimo, de la misma manera se presentan las operaciones posibles de realizar con estos a través del manejo de la estructura bajo los tres niveles de organización a los que se encuentra sujeta.

A través del análisis de estos procesos y funciones podemos entender mas a fondo los objetivos y tareas realizadas por la aplicación, dentro del esquema planteado desde un principio a través de la explicación del funcionamiento e implementación realizada acerca de las funda-

mentos teóricos que sustenta esta aplicación, conjuntamente con los principios de programación orientada a objetos, y la forma en como se involucran las diferentes propiedades asociadas.

Por ultimo cabe hacer mención que a través de la explicación del funcionamiento, así como de la exposición del diseño y de los principios de construcción utilizados para el desarrollo e implementación de las funciones, es posible realizar la siguiente consideración : que dentro de los módulos y funciones necesarios para la implementación de la función de dispersión; y algunos otros casos dentro de los cuales la inclusión dentro de la construcción de funciones de este tipo de elementos de apoyo brindan, mejores elementos que redundas en el aumento y ampliación de las capacidades dedicadas en este caso a la consulta e interacción de sistemas de información, los cuales impliquen el manejo de grandes volúmenes de información, como los sistemas de multimedia, bases de datos de misión critica, y bases de datos de consulta a nivel nacional e internacional.

5.1 Implementación de la función de dispersión

Con fundamento en el análisis que se realizó dentro del pasado capítulo con respecto al diseño de la función de transferencia, partimos de la base de considerar modelos de funciones de dispersión conocidas e implementadas por diferentes autores; estas funciones fueron publicadas en diferentes textos que abordan el tema de la dispersión; así como de textos encontrados dentro de Internet que tratan acerca del tema.

La función de transferencia que se ha diseñado se encuentra establecida con la finalidad de satisfacer los siguientes principios fundamentales :

- ✓ Una máxima capacidad en el manejo de una amplio número de elementos (tamaño del código).
- ✓ Un índice mínimo de incidencia de colisiones, resultado del proceso de dispersión; mediante la aplicación de funciones y operaciones matemáticas simples, mediante uso de exponenciales, logaritmos y multiplicaciones y

diferencias.

- ✓ El manejo de una longitud (máxima) considerable de la cadena mediante la cual se manejaran internamente los procesos involucrados

5.1.1 Implementación de la función

Para esta descripción al igual que la que se realizará con cada uno de los temas a continuación, partiremos de una visión general hacia los aspectos particulares de cada uno de los procesos; por lo que dentro de esta primera sección empezaremos observando con detenimiento las clases que integran el nucleo de la función de dispersión.

Primeramente tenemos la clase dec2bin, la cuál desarrolla la función de convertir un valor de decimal a binario, esta función debido a que se encuentra declarada como de uso público dentro de otra clase, es necesaria que se de defina primeramente, esta clase utiliza variables propias para su desarrollo : índice (int); número (long); *string (char). De forma complementaria se interrelacionan las funciones [ReasignaMem(void)] y [Conversion(void)], las cuales tienen por objeto llevar a cabo la reasignación de memoria para la aplicación de los procesos utilizados dentro de la clase, y la función de conversión de los valores obtenidos de la función de dispersión.

Asi mismo se tienen las siguientes funciones declaradas como funciones públicas, para el uso de la clase: constructores de la clase,[dec2bin()]; (parametrizado) [dec2bin(long valor)]; destructor de la clase, [~dec2bin{ delete[] string; }]; función de conversion de valor decimal a binario, [void Asigna Num(long valor)]; ordena el valor convertido a binario [void espejo(void)]; devuelve el valor transformado en binario, [char const *NumeroBin(void) const];completa el tamño de la palabra, según los bits utilizados en la conversión, [unsigned const NumeroBits(void) const].

Dentro de la clase Dispersión se encuentra definida como pública la clase [dec2bin], asi mismo se definen como variables internas el valor obtenido, [float Valor]; la longitud de la cadena, [int lstring]; manejador de los decimales, [float Decimal]; apuntador de la cadena de

caracteres. De igual forma se tienen de manera interna las funciones de procesamiento, [void Proceso()]; primera parte de la operación de dispersión, [double Suma_ASCII()]; comparación y obtencion de los valores de los caracteres conforme al código del apéndice B [double NvCódigo(int Ctr)], de igual forma se tienen la obtención de los valores [double OBtenNm(double Ptr)]; se calcula el valor de dispersión de la cadena [void Calcula()]

Como funciones publicas se tienen las asociadas a los constructores [Dipersión()] [Dispersión(int Ndecimal, char *Tstring)]; así como el destructor[~Dispersión() {delete string}] ,de igual forma se tiene la función para la asignación de los valores [void AsignaV(int Ndecimal, char *Tstring)],la función que nos permite obtener el numero, en valor decimal [long const NumeroObt() const;], esta nos proporciona el número en binario para el procesamiento [char const *NumeroFn() const]

Primeramente se procede a pasar como parámetros la variable del objeto, con la información necesaria : número de decimales y la cadena a convertir; con los cuales se comienza realizar la operación de inicialización de variables, así como el control del número de decimales con los cuales se va a calcular la dispersión de los elementos.

Posteriormente dentro de la segunda parte del objeto, el algoritmo se basa en la detección de la posición de cada uno de los elementos de la llave, a los cuales se les aplica una operación basada en el número e para aquellos elementos que por su posición sean pares, y una semejante con aplicación del número p para todos aquellos que sean nones, bajo la forma siguiente :

```
doble Dispersión :: Suma_ASCII() {
    double suma = 0.0;
    int flag = 0

    for(int i = 0;i lstring; i++) {
        if(!flag)
            suma+=sqrt(pow(M_E, NvCodigo(String[i ) ) ), flag = 1;
        else
            suma+=sqrt(pow(M_PI NvCodigo(String[i ) ) ), flag = 0;
    }
    return suma;
}
```

Con lo cual se obtiene una cantidad que se acumula en la variable suma, dentro de la cuál se obtiene el valor total de la llave ingresada en la función de dispersión, mediante la adición cíclica de cada uno de los valores obtenidos a través de la función NvCodigo, la cual opera de manera simultánea para la obtención de los valores de los caracteres, según tabla anexa en el apéndice a, donde se precisan los símbolos, así como su valor en código ASCII, y su valor en código alterno calculado.

```
doble Dispersión::NvCodigo(int Ctr) {
    int Cvlor = 0;

    iff( int (Ctr) == 32 ) Cvalor = 99;
    else if ( 33 <= int(Ctr) && int(Ctr) <= 7 )
        Cvalor = int (Ctr) - 32;
    else if ( 65 <= int(Ctr) && int(Ctr) <= 90 )
        Cvalor = int (Ctr) - 39;
    else if ( 97 <= int(Ctr) && int(Ctr) <= 122 )
        Cvalor = int (Ctr) - 45;
    else Cvalor = 0;

    return double(Cvalor/10.0);
}
```

Acto seguido el valor resultante de la variable suma, obtenida en la función Suma_ ASCII(), se le obtiene el logaritmo natural; y posteriormente se multiplica por el resultado del logaritmo base diez del número obtenido hasta aquí , elevandolo despues a la longitud de la cadena. Se procede a obtener el módulo flotante; mediante la realización de la diferencia de la parte real y la parte entera, y el valor obtenido se multiplica por el número de decimales solicitado, y se almacena en la variable Valor; así mismo se vuelve realizar este proceso y a la parte resultante se le vuelve a sacar el módulo flotante y es depositado en la variable Ivalor la cual posteriormente es asignada a la variable Valor.

```
void Dispersión::Proceso() {
double Ivalor =0.0;

Valor = log ( Suma_ASCII() ) * log10 (pow ( M_PI , lstring );
modf(Valor,&Ivalor);
Valor -= Ivalor; Valor *= Decimal;
modf(Valor,&Ivalor);
Valor -= Ivalor, Valor *= Decimal;
modf(Valor,&Ivalor);
Valor = Ivalor;
```

```
}
```

El resultado de la operación asociado a la variable Valor, es el producto final que se desprende de la función de dispersión antes de la conversión del número de base decimal a base binaria.

El proceso de conversión del valor obtenido a través de la función de transferencia se lleva cabo a través de la función conversión, la cuál es una función recursiva, ya que cuando entra el valor a esta es asignado a la variable número, en el proceso AsignaNum(), y posteriormente pasa al proceso Conversión (), el cuál detecta si es un numero mayor o igual a 0 y/o 1, en caso de ser mayor, se procede dividir entre dos e ir obteniendo los resultados parciales, estos con el fin de concatenar al resultado un cero o uno según se vaya obteniendo dentro del resultado de la división; posteriormente se vuelve a dividir el número entre dos, lo cual provoca como consecuencia que se incremente el índice de recorrido del número y invoque nuevamente a la función, hasta obtener como resultado 0 ó 1 y finalizar el proceso.

```
void dec2bin::Conversión() {
 if( !numero || numero==1 ) {
    if( !numero )
       strcat(string,"0");
    else
       strcat(string,"1");
    Espejo();
    return;
 }
 else {
    if(!(numero %2))
       strcat(string,"0");
    else
       strcat(string,"1");
     numero /2 , indice++;
    ReasignaMem();
    Conversión();
 }
}
```

Esta operación se complementa pasando a través de un procedimiento espejo, que invierte la cadena obtenida dentro de la conversión de base, con el fin de poder almacenar el numero binario de forma correcta, ya que al momento de la conversión se concatena en forma contraria.

```
void dec2bin::Espejo() {
```

```
    char *temp = new char [indice ;
    int j = 0; temp [j] = '/0';

    for( int i = indice-1; i -1; i—)
        (char) temp [j] = chr string[i] , j++;

    strcpy(string,temp)
    string[indice] = '\0';
    delete temp [];
}
```

Una vez que se ha realizado la función espejo antes mencionada; el valor de la variable string es devuelto a través de la variable de objeto *Dispersion*::NumeroFn(); la cual retorna la cadena obtenida dentro del proceso del objeto *denominado* NumeroBin().

5.2 Implementación del árbol dinámico

En lo que se refiere a la estructura de datos necesaria para el manejo del almacenamiento de información dentro de la memoria, durante la ejecución de la aplicación; se ha tomado en cuenta una técnica, la cual se encuentra basada en la aplicación de mecanismo que permite administrar la utilización de los espacios disponibles dentro de la misma, lo anterior se realiza mediante la creación de una estructura de un árbol dinámico, que permita llevar a cabo un control eficiente de las paginas, lo anterior fundamentado a través del uso de un concepto denominado factor de carga, el cual nos permite llevar a cabo un mejor aprovechamiento de la capacidad total ocupada dentro de la página.

Dentro de la clase denominada Node se encuentran *definidas como* publicas aquellas funciones asociadas a la destrucción del nodo(virtual ~Node();), verificación de la duplicidad del nodo (virtual Node* duplicate(void) const = 0;), comparación de dos nodos (virtual int compare(Node const *other) const = 0;), operación sobre los datos que se encuentran dentro del nodo (virtual void process(void) = 0;), y verificación de preexistencia del nodo durante la adición de un nodo dentro del árbol (virtual void already_stored(void);)

```
class Node
{
    public :
    // destructor
    virtual ~Node();
```

```
    // duplicador
    virtual Node* duplicate(void) const = 0;

    // comparación de dos nodos
    virtual int compare(Node const *other) const = 0;

    // función que opera sobre los datos contenidos dentro de un nodo
    virtual void process(void) = 0;

    // Es llamada cuando un nodo es sumado en el árbol
    virtual void already_stored(void);
};
```

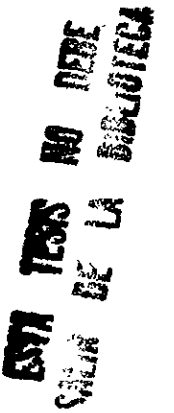
Las funciones del destructor, así como del verificador de preexistencia, son funciones similares, en tanto de que una se encarga de la liberación de los recursos ocupados, y de no operar mas sobre la incidencia mencionada, la función de pre-existencia, se ocupa de no ejecutar ninguna operación, en el caso de que la función de duplicado haya regresado un valor de verdadero.

```
Node::~Node(){ }

void Node::already_stored() { }
```

En lo referente a la clase Tree en donde se define al árbol como una estructura de datos existente dentro de la memoria, tenemos definidas de forma publica dentro de la clase las siguientes funciones : el destructor del árbol(~Tree(void);), los constructores (Tree(void);Tree(Tree const &other);), asignación de elementos del árbol (Tree const &operator= (Tree const &other);), adición de un nodo (void add(Node *what);), los recorridos posibles bajo diferentes opciónes de orden dentro del árbol (void preorder_walk(void);void inorder_walk(void);void postorder_walk(void);).

Asimismo de forma privada para uso exclusivo de la propia clase tenemos las siguientes funciones: copiar (void copy(Tree const &other);), destruir (void destroy(void);), asi como tambien tenemos ciertos datos que se usan de manera exclusiva para el manejo del árbol que son : Tree *left, *right, que son apuntadores de indicación de las ramas, así como Node *Infor, el cual es un apuntador directo a la información que se encuentra almacenada dentro de un nodo en especifico.

```
class Tree {
    public :
    // destructor, constructores
    ~Tree(void);
    Tree(void);
    Tree(Tree const &other);

    // asignación
    Tree const &operator= (Tree const &other);

    // adición de un nodo
    void add(Node *what);

    // procesando el orden en el árbol
    void preorder_walk(void);
    void inorder_walk(void);
    void postorder_walk(void);

     private :
    // primitivas
    void copy(Tree const &other);
    void destroy(void);

    //datos
    Tree *left,*right;
    Node *info;
};
```



El destructor en este caso tiene la función de tomar el recurso asociado al proceso y liberarlo de manera inmediata, ya que no va a ser utilizado, en este caso en particular se utiliza el comando destroy(), el cual procede de inmediato a la liberación de la estructura del árbol que se encuentra implementada dentro de la memoria.

```
// destructor : destruye el árbol
Tree::~Tree() {
  destroy();
}
```

Al momento en que se inicia la clase este constructor procede a inicializar los parámetros necesarios para la construcción y manejo de la parte dinámica del árbol, inicializando los indicadores de las ramas (left, right) en cero, de igual manera la bandera indicador de almacenamiento de información se inicializa en cero.

```
// constructor por default: inicializando en cero
Tree::Tree() {
    left = right = 0;
    info = 0;
}
```

Este constructor se utiliza, en el caso de tener que llegar a la eliminación de un nodo y cuando se utiliza el comando guardar, es cuando los nodos ya tienen asignada información dentro de ellos, por lo que se procede a la destrucción del árbol, el cual se vuelve a construir, y se le reasigna la información contenida dentro de cada uno de los nodos, a la nueva estructura del árbol.

```
// constructor copia : inicializa los contenidos de otros objetos
Tree::Tree(Tree const &other) {
    copy(other);
}
```

Para el caso de cuando se añade o se contrae una rama del árbol, se utiliza el siguiente constructor, el cual destruye la estructura del árbol utilizado hasta el momento, y realiza un rastreo de todos los subárboles contenidos dentro de la memoria, verifica los contenidos de los nodos, los copia y procede a reagruparlos dentro de una nueva estructura, según los contenidos de información de cada nodo.

```
// asignación sobercargada
Tree const &Tree::operator= (Tree const &other) {
 if(this != &other)  {
  destroy();
  copy(other);
 }
 return (*this);
}
```

Este proceso describe la adición de un nodo dentro del árbol, este inicialmente verifica que la información no se encuentre duplicada, en el caso de ser así, no inserta el nuevo nodo; por el contrario, cuando el resultado es negativo, procede a crear la liga con el nodo precedente; de igual forma, diagnostica en que lado es necesaria la adición, es decir si adiciona dentro de la rama izquierda o derecha, según las ramas o nodos que hayan sido añadidas previamente al mismo, y ya que se han identificado, realiza la adición del nodo y procede a la liberación de los

recursos para la copia de la información a ingresar dentro del mismo.

```
// sumando un nuevo nodo al árbol
void Tree::add(Node *what) {
 if(!info)
   info = what-duplicate();
 else  {
  register int
  cmp = (int) info-duplicate();
  if(cmp) {
   if(!left) {
       left = new Tree;
       left->info = what-duplicate();
   }
   else
       left-add(what);
  }
  else if(cmp) {
   if(!right)  {
       right = new Tree;
       right-info = what->duplicate();
   }
   else
       right-add(what);
  }
  else
    info-already_stored();
 }
}
```

La función que se tiene a continuación, realiza un recorrido dentro del árbol a través de un algoritmo denominado preorder, el cual funciona de la siguiente manera:

- ✓ Se consulta el contenido del nodo raíz, y en el caso de encuentra alguna información se consulta y/o procesa.
- ✓ En el caso de que no encuentre información, busca primero dentro del nodo de la rama izquierda, y en caso de localizar algún contenido, procede a la consulta y/o procesamiento del mismo.
- ✓ Posteriormente se realiza el procedimiento anterior dentro del nodo perteneciente a la rama derecha.

```
// implementación de búsqueda en algoritmo " preorder " dentro del árbol

void Tree::preorder_walk() {
 if(info)
  info->process();
 if(left)
  left->preorder_walk();
 if(right)
  right->preorder_walk();
}
```

La función que se presenta a continuación, realiza un recorrido dentro del árbol a través de un algoritmo denominado inorder, el cual se realiza la siguiente forma :

- ✓ Primeramente busca la información primeramente dentro del nodo situado en la rama izquierda, y se procede a la consulta y/o procesamiento del contenido.
- ✓ Retorna al nodo raíz o precedente anterior, y procede a realizar la consulta y/o procesamiento de la información que encuentra dentro de el.
- ✓ Finalmente realiza la búsqueda hacia la rama derecha, y en cuanto encuentra la información la consulta y/o procesa.

```
// implementación de búsqueda en algoritmo " inorder " dentro del árbol

void Tree::inorder_walk() {
 if(left)
  left->inorder_walk();
 if(info)
  info->process();
 if(right)
  right->inorder_walk();
}
```

Este mecanismo en especifico, su funcionamiento se manifiesta de la siguiente forma:

- ✓ Se toma la rama izquierda y la recorre toda, siguiendo la tendencia, una vez que llega al ultimo nodo; consulta y/o procesa la información

- ✓ De igual forma una vez que se ha finalizado con la rama izquierda, se procede a la visita la rama derecha, y en el caso de contener información se consulta y/o procesa.

- ✓ Conforme termina los recorridos por las ramas regresa a verificar el contenido del nodo padre, y en el mismo realiza la consulta y/o procesamiento de los anteriores.

```
// implementación de búsqueda en algoritmo " postorder " dentro del árbol

void Tree::postorder_walk() {
 if(left)
    left->postorder_walk();
 if(right)
    right->postorder_walk();
 if(info)
    info->process();
}
```

Dentro de esta función se observan las instrucciones destinadas a la destrucción de los elementos que controlan el árbol, lo anterior con la finalidad de liberar la estructura y los recursos usados para su implementación, se procede al borrado de la información del nodo raíz, y conjuntamente se eliminan de igual forma los apuntadores que funcionan como ligas de interrelación, con las ramas izquierda y derecha.

```
// operaciones primitivas

void Tree::destroy(){
 delete info;
 if(left)
    delete left;
 if(right)
    delete right;
}
```

En este caso la función principal de la función es la de realizar el proceso de copia de la información, verificando la duplicidad de información que se encuentra contenida entre los diferentes elementos de información del árbol; este proceso se realiza en tanto se lleva a cabo la

eliminación del árbol anterior y la copia de la información dentro de la nueva estructura. Se comienza por verificar la duplicidad de la información, y se verifica rama por rama, comenzando para la izquierda y después hacia la derecha, con la finalidad de crear una copia exacta de las conexiones dentro de la nueva estructura.

```
void Tree::copy(Tree const &other) {
 info=other.info ? other.info->duplicate() : 0;
 left=other.left ? new Tree(*other.left) : 0;
 right=other.right ? new Tree(*other.right) : 0;
}
```

Las anteriores clases por si mismas no logran desempeñar, toda la compleja funcionalidad de la operación del árbol dentro de la memoria, razón por la cual se hace necesario el uso de una clase auxiliar, la cual nos permita implementar el manejo de las funciones a un nivel mas detallado, para el caso las clases Node() y Tree() manejan los parámetros globales y dan seguimiento a cada uno de sus elementos componentes como un todo. Es decir la clase Node maneja los aspectos relativos al nodo como entidad de almacenamiento de información, de igual forma la clase Tree se encarga de manejar el árbol creado dentro de la memoria como una estructura de datos particular, proporcionando el control de las operaciones que lo afectan como entidad de control.

La clase que se presenta a continuación denominada Strnode, es la clase que se encarga de la aplicación de los procesos que se encuentran inmersos dentro de la clase Node(), cuyo funcionamiento y desempeño se encuentra vinculado a las operaciones desarrolladas con la información contenida dentro de cada uno de los elementos que puede almacenar un nodo.

Dentro de la definición propia de la clase para su uso publico se tiene la clase Node, así como también se tienen las funciones: ~Strnode(void), el destructor de la clase; y los constructores Strnode(void); Strnode(Strnode const &other); Strnode(char const *s); de igual manera se realiza una asignación para manipular el uso de los contenidos de la información; Strnode const &operator= (Strnode const &other); de igual forma se requiere el uso de algunas funciones de la clase Node() aplicando la propiedad de funciones fraternas, y estas son : Node* duplicate(void) const; int compare(Node const *other) const; void process(void); void already_stored(void).

Dentro de lo que se refiere a la parte privada, encontramos la definición de algunas variables, para la realización de procesos internos : char *str; int times.

```
// Aplicación de las clase Node() ya antes editada

class Strnode : public Node {
 public :

 // destructor
 ~Strnode(void

 // constructores
 Strnode(void);
 Strnode(Strnode const &other);
 Strnode(char const *s);

 // asignación
 Strnode const &operator= (Strnode const &other);

 // funciones que requieren usar el protocolo de la clase Node
 Node* duplicate(void) const;
 int compare(Node const *other) const;
 void already_stored(void);

 private :

 //datos
 char *str;
 int times;
};
```

A continuación tenemos el destructor de la clase, el cual elimina y libera los recursos relativos a la cadena que maneja los contenidos de información de los nodos.

```
Strnode::~Strnode() {
 delete str;
}
```

El constructor inicializa los valores respectivos a la cadena y el contador str y times, así como también corrobora la disponibilidad de los recursos para ser utilizados, por la clase Strnode()

```
Strnode::Strnode() {
 str=0;
 times=0;
}
```

Este constructor, se utiliza cuando se lleva a cabo la tarea de copiado de un elemento contenido dentro de un nodo y que va hacia otro nodo, esto debido a la adición y contracción de elementos, es dentro de este constructor, en donde es posible manejar propiedades de valores heredados, en este caso los valores other.str y other.times de la función strdup(), con el objeto de no provocar la duplicación de elementos de información pre-existentes dentro del árbol.

```
Strnode::Strnode(Strnode const &other) {
 str = strdup(other.str);
 times = other.times;
}
```

Este constructor en especifico hereda los valores de inicialización provenientes de la clase Node, con el fin de llevar a cabo la construcción del árbol y la copia de los elementos de información contenidos, para inicializar el árbol de datos.

```
Strnode::Strnode(char const *s) {
 str = strdup(s);
 times=1;
}
```

Dentro de esta función se aplica sobrecarga al operador, con la finalidad de realizar la copia del elemento, evitando la duplicación de la información, se elimina la anterior liga de Str y se copian los valores provenientes de la función strdup(other.str); en tanto que el valor de other.times, es heredado para llevar a cabo la copia dentro de la posición necesaria del árbol.

```
Strnode const &Strnode::operator= (Strnode const &other){
 if(this != &other) {
  delete str;
  str= strdup(other.str);
  times=other.times;
 }
 return (*this);
}
```

En el caso de que la información se encuentre duplicada, se crea un nuevo objeto, el cual es asignado dentro del apuntador *this y este procede a regresar el valor a la función que lo solicito, en este caso la función que desarrollo la comparación t copia de la información, para la renovación de la estructura.

```
Node *Strnode::duplicate() const {
  return (new Strnode(*this));
}
```

En esta función se involucra como parámetro, el apuntador, el cual contiene información copiada de la estructura anterior del árbol de datos, verifica que la información no se encuentre duplicada, en el caso de que asi sea, retorna un valor de 0 a la bandera, así como la información que se ha encontrado duplicada .

```
int Strnode::compare(Node const *other) const {
  Strnode
  *otherp = (Strnode *) other;
  if(str && otherp-str)
    return (strcmp(str,otherp-str));
  if (! str && ! otherp-str)
    return (0);
  return ((int) otherp-str - (int) str);
}
```

En el caso de que el valor, ya haya sido previamente almacenado en el lugar indicado, el contador times se incrementa uno mas el valor actual, con el fin de que se continúe la copia del siguiente elemento que sea necesario.

```
void Strnode::already_stored() {
  times++;
}
```

5.3 Referencias

[1] Schildt, Herbert. "C++ : Guía de autoenseñanza". McGraw Hill. México. 1995.

[2] Frank B. Brokkenend and Karel Kubat. "C++ Annotations Versión 4.2.4".ICCE, University of Groningen Westerhaven 16, 9718 AW Groningen. Netherlands Published at the University of Groningen. ISBN 90 367 0470 7.1994 - 1997.

[3] Chris H. Papas and William H. Murray. "Manual de Borland C++ 4.0". Mc Graw Hill. España. 1994.

[4] Coad, Peter and Yourdan, Edward. "Object - Oriented Analysis". Secon Edtion. Prentice Hall Building. N.J.. U.S.A.. 1990

Capítulo 6

Pruebas

Las estructuras de datos son el soporte principal para el desempeño de cualquier aplicación que pretenda procesar grandes volúmenes de información. Por citar algunos ejemplos : bases de datos, hojas de calculo, procesadores de texto, empaquetadores de información, etcétera. Así, cada una de las aplicaciones antes mencionadas ocupa una estructura de acuerdo a la necesidad que deba solucionar.

Para tal fin es necesario desarrollar nuevas alternativas de las estructuras, con la finalidad de mejorar el desempeño de las aplicaciones. Este el caso en específico de la dispersión dinámica virtual (DDV). Ya en anteriores capítulos se realizó un análisis detallado con su respectiva implementación, sin embargo ha llegado el momento de verla funcionando dentro de una de las aplicaciones antes citadas.

Para comprender mejor el uso de esta técnica, será necesario observar el tipo de aplicación que se desarrolló, y cual es su funcionamiento dentro de la misma. Para después ver las pruebas que aplican, así como los resultados obtenidos dentro del desempeño.

6.1 Criterios de selección

La DDV se puede aplicar en diversas funciones, las cuales se encuentran dentro de un amplio contexto de aplicaciones. Algunas ofrecen un alto grado de complejidad como lo son las bases de datos y las hojas de calculo. Otras requieren un uso más extenso del procesador, siendo el ejemplo más claro de este tipo de aplicaciones los empaquetadores de archivos y los manipuladores de gráficos. Para ilustrar mejor el desempeño de la DDV, se determino un procesador de textos básico. El decir básico se refiere a aplicarla en un modelo didáctico, en este caso a que debe cumplir con las siguientes características :

- ✓ Manejo de archivos (crear, abrir, cerrar, guardar e imprimir).
- ✓ Opciones de edición (cortar, copiar, pegar y deshacer la última operación).
- ✓ Opciones de búsqueda (buscar, buscar y reemplazar).

A los anteriores objetivos se le adicionan los siguientes :

- ✓ Óptimo manejo de la memoria. Tanto para el procesamiento de los documentos como la capacidad de almacenamiento en la misma.
- ✓ Velocidad de procesamiento, que no depende de la cantidad de memoria que se tenga, sino de la capacidad de la estructura ante un significativo número de documentos como de su tamaño.

6.2 Estructura

La DDV se esquematiza en la figura 6.1, dentro de la cual podemos visualizar los 3 elementos que la componen. El primero de ellos comprende a la función dispersión, la cual se integra de forma estructural con el árbol, cuyo esquema corresponde a la un tipo radix en base 2^1, el cuál es siguiente elemento dentro de la figura; mientras que el tercer elemento, o se el manejador de la memoria virtual se acopla no solo al árbol, sino que tambien adopta la modalidad de poder interactuar de una manera más segura con los requerimientos de la aplicación y del sistema operativo (SO). Además se puede apreciar que una parte considerable de las tareas realizadas por este bloque se desarrolla a través del manejo del árbol, pues ahí se desarrolla la administración de la información con apoyo de la función de dispersión. Para poder realizar estas operaciones debe de contar con los suficientes recursos de memoria, por lo que recurre al bloque de la memoria virtual para incrementar más recursos y poder cumplir con su tarea establecida.

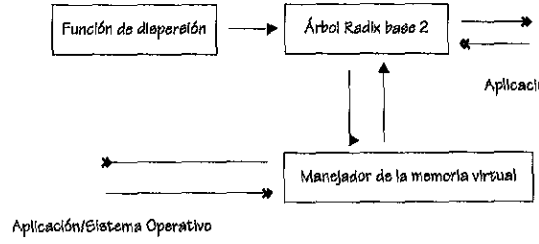


Fig. 6.1 Diagrama de bloques de la DDV.

El bloque de la aplicación que se ilustra en la figura 6.2, a través de este se pueden apreciar los

1 Para mayor referencia ver el capítulo 4.

distintos elementos que integra la operación de la DDV dentro del mismo programa. Dentro de este esquema se puede observar que hay tres bloques en el mismo nivel : *Edición, Búsqueda* y *Archivos*. Los cuales interactuan sobre los documentos que se cargan en la aplicación, siendo una de sus principales características del programa es la de que se pueden trabajar más de un documento a la vez, es decir, es una aplicación MDI (por sus siglas en ingles, *Multiple Document Inteface* - Interfaz Multiple de Documentos).

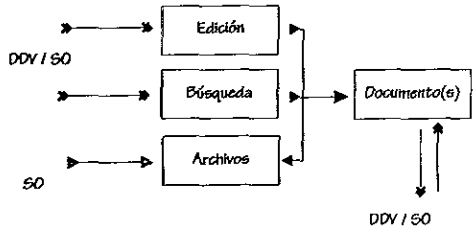


Fig. 6.2 Diagrama de bloques de la aplicación.

Los bloques de *Edición, Búsqueda* y *Documento(s)* interactuan con la DDV y el sistema operativo (SO), dado que los tres bloques necesitan de recursos de memoria para realizar sus tareas, y a la vez de enviar la información que ha sido modificada por los otros bloques; para poder mantener actualizado el texto que se mantiene activo. Mientras que el bloque *Archivos* trabaja sobre los documentos con ayuda del SO para realizar las tareas básicas del manejo de archivos y de impresión.

6.2.1 Forma de trabajo

Cada uno de los bloques realiza tareas muy especificas dentro del programa, para lo cual es necesario detallar más su acción. Para el caso de las anteriores figuras, se mostró de modo general su interrelación dentro del programa, y se partió de la presentación de un algoritmo de innovación a la aplicación. Ahora se considera conveniente ilustrar el funcionamiento de cada uno de los bloques. Comenzando a partir de la figura 6.2. El desarrollo se hará de derecha a izquierda para llevar una secuencia más clara.

El primer bloque, que se denomina ***Documentos(s)***. En él se desarrollan las siguientes tareas :

- ✓ Una vez que el bloque ***Archivos*** envía la dirección donde se ubica el documento a procesar, se establecen los parámetros de inicio. Los cuales definen

la cantidad inicial de recursos de memoria para el manejo del documento a través del árbol. Esta cantidad se determina de la siguiente manera : Primero se realiza un chequeo de la cantidad de memoria física ocupada hasta ese momento por otras aplicaciones que se encuentran en ejecución y además por los documentos que se encuentren ya previamente cargados en el programa. Sí la información que se obtiene demuestra que la cantidad de memoria física ya es insuficiente, se recurre a la DDV/SO para otorgar la memoria faltante a través de la memoria virtual. Esto se hace con la finalidad de que el documento a procesar encuentre la cantidad necesaria de recursos de memoria para trabajar. Este punto se también aplica para el caso de que requiera crear un nuevo documento utilizando esta aplicación, pero con la siguiente observación, que siendo un documento nuevo se asigna una cantidad X de memoria ya sea física y/o virtual.

- ✓ Las actualizaciones que se realizan en el bloque de ***Edición*** y en el de ***Búsqueda***, se ven reflejadas sólo en la memoria física/virtual donde se encuentra el documento. Lo anterior es para no tener que recurrir al disco duro, almacenarlo y luego presentarlo, pues implica realizar varios accesos al mismo, con el fin poder actualizar la información contenida en el documento.

- ✓ Así mismo bloque de ***Archivos*** se encarga además de enviar las acciones de cerrar y guardar el(los) documento(s) en la aplicación. Sí lo desea el usuario puede almacenar las acciones que hayan modificado el contenido del archivo; En tal caso, el árbol regenera en el archivo y espera nuevas instrucciones. Liberando memoria y adecuandonse al nuevo contenido. Mientras que en el caso de cerrarlo, se liberan por completo los recursos de memoria utilizados.

Para el caso del bloque de ***Edición*** tenemos lo siguiente :

- ✓ En el se desarrollan las acciones básicas de copiar, pegar y cortar ya sea letras, palabras o párrafos inclusive. Las dos últimas acciones repercuten dentro del bloque ***Documento(s)*** pues es en este, donde se encuentra la información a procesar. En el caso de pegar, se suman más palabras al árbol,en tanto en el

caso de Cortar, se le eliminan palabras. Estas acciones hacen que se expanda o se contraiga el árbol.

- ✓ Además, aquí se realiza la inserción de información al documento. Logrando con esto la expansión del árbol.

Para el caso del bloque de ***Búsqueda*** , sus principales acciones son las siguientes :

- ✓ Este bloque se divide en dos partes, la primera de ellas se encarga de realizar una búsqueda en documento. Esta acción no incide en la información contenida en el documento. Pues sólo muestra las ocurrencias encontradas en el documento.

- ✓ La segunda acción si incide de manera directa en el documento, pues aparte de realizar la búsqueda, una vez que encuentra la ocurrencia solicitada procede a la modificación. Logrando con esto que el árbol que contiene al documento se contraiga o se expanda de acuerdo a las modificaciones propuestas.

Estas acciones resumen de manera breve los procesos por cada uno de ellos. En el caso de ***Archivos***, como se pudo notar se realiza una explicación a lo largo de los otros bloques, especialmente en el de ***Documento(s)*** pues en este, donde incide de manera directa más que en los otros. De tal manera, la aplicación queda de la siguiente forma :

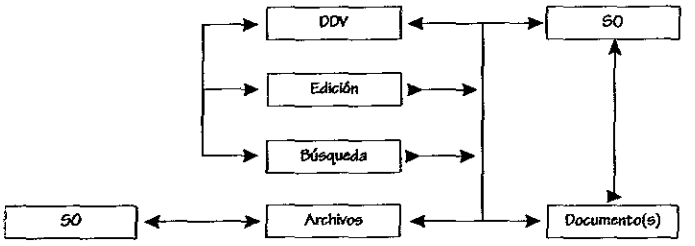


Fig. 6.3 Diagrama final de la aplicación.

6.3 Pruebas de desempeño

Para poder medir el desempeño de la aplicación, siendo este el caso del procesador de palabras, como lo es este, es necesario recurrir a una serie de pruebas. Cada una de estas pruebas se basan más en la velocidad y la capacidad de procesamiento, y el manejo de ambas situaciones en casos extremos.

6.3.1 Dispersión

Las pruebas de dispersión se realizan con lotes de archivos de diferente tamaño y con una variable extra : un índice que determina el número de dígitos a discriminar una vez obtenido el valor final[2]. Para ilustrar lo anterior tenemos la siguiente tabla :

	2	3	4
acera	32	218	1809
base	21	190	9018
copiar	69	923	2314
doble	68	879	7960
excepto	55	582	8263

Tabla 6.1 Valores obtenidos por el manejo de decimales.

Así tenemos que para la palabra *acera* y con un índice = 2, su valor de dispersión es el 32. Cuando es 3, su valor es el 218; y cuando es 4, su valor es el 1809. El asunto aquí el proceso comienza a complicarse de manera substancial, pues se requiere más recursos de memoria para el manejo de la palabra cuando el índice es mayor que 2. Dado que el manejo de los números se realiza en binario, tenemos que para la misma palabra con un índice = 2, tenemos que el valor es de 100000_2, mientras que para 3 el número de dispersión es el 11011010_2. Como se puede observar, se emplea un número mayor de elementos para lograr una mejor dispersión,

2 Para mayor referencia ver el capítulo 4.

pero implica también la inversión de una cantidad mayor de recursos de memoria, como puede observarse cuando el índice es 4 y el valor de dispersión es 10001000001_2.

Para ilustrar lo anterior tenemos la siguiente gráfica, en la cual el índice de dispersión (*I*) es igual a dos para un archivo que contiene 250 palabras diferentes. Se puede apreciar que el nú-

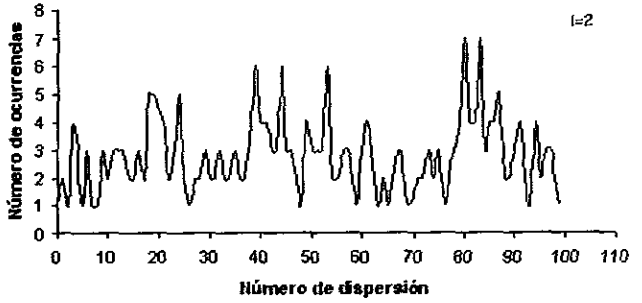


Fig. 6.4.1 Valores de dispersión con índice=2.

mero de datos dispersos encuentra un número considerable de colisiones y que se encuentra en un nivel bastante alto de oscilación. Ahora bien, si observamos la figura 6.4.2 veremos que el nivel de colisiones desciende considerablemente pero sin ser todavía el nivel requerido para el manejo de información.

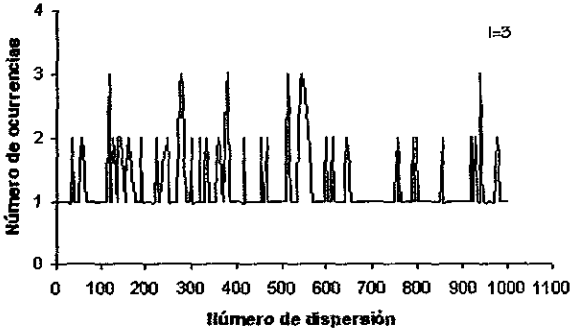


Fig. 6.4.2 Valores de dispersión con índice=3.

En este caso, el nivel de oscilación no alcanza valores arriba de tres, pero aún así presenta una oscilación que hace que la función de dispersión no sea la deseable. Pero en cambio, la figura 6.4.3 nos muestra un nivel bastante aceptable para poder manipular mejor los valores de dispersión obtenidos, pues el nivel de dispersión con *I*=4, presenta una gráfica mucho más estable. Existen todavía algunas colisiones, pero como se menciono antes, pueden procesar de una manera rápida y sin tantos conflictos.

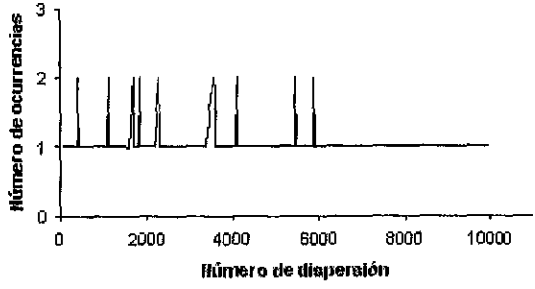


Fig. 6.4.3 Valores de dispersión con índice=4.

Aquí sobresale un pregunta interesante : Sí la primera gráfica ocupa un *I*=2 y un archivo de 250 palabras ¿No debería ser menor el archivo menor o igual a 100 palabras ? La respuesta es

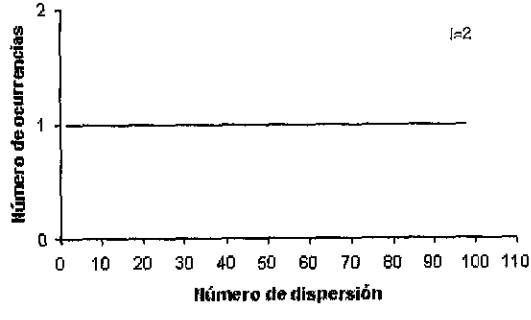


Fig. 6.4.4 Valores obtenidos de un archivo menor y con un índice=2.

sí. La figura 6.4.4 nos muestra lo que pasa cuando un archivo es menor a 100 palabras y con un índice igual a 2. Se presento el caso en el que el archivo contiene un número mayor de palabras que las que se pueden trabajar. Si se mantiene el índice fijo y es creciente el número de palabras en el documento, la figura 6.4.1 es el resultado que se obtiene, esto nos implica un nivel de complejidad bastante alto para resolver las colisiones que se presentan. Por lo tanto se aconseja que se maneje un $I'=I+2$ para un manejo óptimo de la información.

6.3.2 Accesos a disco

Los accesos a disco es una de los aspectos más relevantes que se tomaron en cuenta para el desarrollo de la DDV. Este factor se califico de acuerdo a la cantidad de memoria física que se tenga, y para este caso en especifico, se utilizó una PC con procesador 486 DX2 a 50 Mhz con un disco duro de 500 Mbytes con Windows 3.11. El único aspecto se que se fue modificando fue el de la memoria RAM. La figura 6.5 ilustra el comportamiento obtenido, pues la aplicación necesita de 4 MBytes en RAM para poder trabajar, que es el punto de inicio en el eje de la abcisas de memoria, mientras que el eje de las coordenadas a disco no puede pasar de 2. Pues una de las premisas es esa, que no exceda este valor por el contrario que tienda a ser el óptimo.

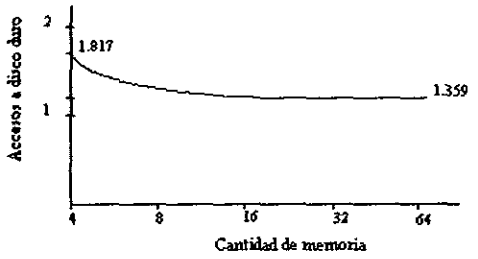


Fig. 6.5 Accesos a disco duro.

En la gráfica se observa un decrecimiento suave entre los 4 y 16 Mbytes, pues a menor memoria mayor es el número de accesos a disco. después de los 16 Mbytes el número de accesos se estabiliza en un valor dado. Este valor no logra descender más por las siguientes razones : primero, no se pretendió crear un manejador externo de memoria virtual para el soporte de Windows; segundo, la aplicación al hacer un uso extenso de la memoria para el manejo de los documentos, requiere mantener constantemente actualizados los movimientos y los documentos por ende.

En el caso de Windows 95 se observan aspectos muy similares el comportamiento. La curva presenta un acentuamiento todavía más suave, pero los valores de asentamiento no cambian mucho (de inicio empieza en 1.825, y se asienta en 1.347). Mientras que el que caso de Windows 3.10 las variaciones no alcanzaron valores que fueran tomados como significativos, pues alcanzaron los mismos que en el caso de Windows 3.11.

6.3.3 Consideraciones acerca del tiempo de paginación

Con fundamento en lo anterior, deseamos aclarar que el Administrador de memoria es propio de la plataforma de Sistema Operativo dentro de la cuál se encuentra ejecutándose la aplicación, es decir: DOS, Windows 3.1, Windows 95 , etc., prácticamente la estructura implementada, realiza los procesos de apoyo y optimización del desempeño del adminisrtrador de memoria, sin embargo el comportamiento del proceso es el propio de la plataforma, por lo que a continuación se presenta una tendencia acerca del comportamiento del tiempo de paginación dentro de los gráficos siguientes.

Como podemos observar dentro de la plataforma de Windows 3.X, conforme aumenta el tamaño de un archivo en Kbytes el tiempo de paginación dentro de un equipo con 4 MB de memoria en RAM, aumenta progresivamente, debido al uso de los bloques de memoria virtual dentro del disco duro, en cambio, conforme a la memoria RAM, si se aumenta a 8, 16, 32, ... MB esta curva de comportamiento tiende a suavizarse y tenderá a cero conforme al tamaño de la memoria disponible, hasta darse el caso de cero,

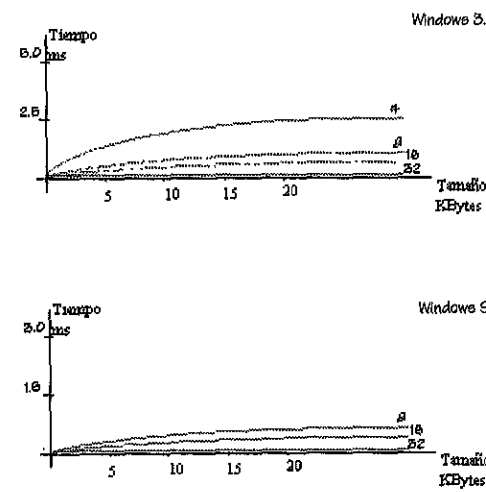


Fig. 6.6 Tendencias del comportamiento del tiempo de paginación..

debido a que la Memoria RAM contiene la totalidad del archivo, ya que su tamaño en referencia a la memoria disponible no es considerable.

De igual forma en lo que respecta a Windows 95, la consideración anterior funciona de igual forma, adicionalmente que los tiempos de acceso son menores, debido a los mecanismos de administración de memoria implementados dentro de la plataforma, se consideran 8 MB, debido a que así fué probado con un equipo, más sin embargo las consideraciones técnicas para el desempeño óptimo de Windows 95 proporcinadas por Microsoft Corp. Lo sitúa en 16 MB.

Sin embargo se presenta un comportamiento similar, a medida que aumenta la memoria RAM disponible dentro del equipo, el usos de la memoria virtual tiende a ser nula.

Capítulo 7

Conclusiones

Con fundamento en los trabajos realizados para la elaboración de este proyecto de tesis; y tomando como guía para el desarrollo de estos, el objetivo que nos dice: " Diseñar los algoritmos de búsqueda, inserción y borrado, para la creación de un algoritmo de naturaleza dinámica que nos permita mejorar el desempeño de una librería de aplicación, en lo relativo a los aspectos de: optimización de tiempos de búsqueda e inserción de elementos de información; asimismo se pretende proporcione beneficios orientados hacia la disminución del numero de accesos necesarios a los dispositivos de almacenamiento secundario ", a partir de un análisis de lo anterior podemos establecer las siguientes conclusiones, por cada una de las partes que integran el desarrollo de este estudio.

7.1 Investigación

Este aspecto en particular, requirió de un esfuerzo de conocimiento; así como de una compenetración plena dentro del tema de la tesis, primeramente a través de una revisión exhaustiva acerca de trabajos de la misma naturaleza, que han sido desarrollados dentro de la propia Universidad y algunas otras instituciones del país, encontrando como resultado que hasta el momento (1994-1997) no se ha desarrollado ningún sistema en especifico enfocado hacia el tema.

Esto nos brindo la capacidad de aportar nuevas ideas y conceptos al tema y a la vez poder ser innovadores dentro del campo (esto a nivel nacional), sin embargo este hecho, nos ha establecido como compromiso llevar a cabo una intensa búsqueda de información dentro de los distintos medios de consulta a nuestro alcance, los cuales nos han permitido establecer las interrelaciones necesarias hacia el tema a desarrollar; consultando desde los libros que nos brindaran elementos para la conformación de la base conceptual, hasta paginas de la Internet que se aborda el tema de manera teórica; así como también se reconoce el apoyo trascendental dentro de este estudio, de algunos profesores de distintas Universidades de Inglaterra, Estados Unidos, Alemania y Australia, los cuales con los conocimientos adquiridos coadyuvaron a su desarrollo.

La etapa de investigación permitió enfocarnos de lleno en el tema, así como también nos apoyo en la delimitación perfecta de los conceptos de la tesis, de igual forma, nos brindo la opor-

tunidad de relacionarnos más ampliamente con personas del propio campo, así como apreciar y posteriormente intentar comprender los elementos utilizados dentro de los diferentes trabajos desarrollados, los cuales nos proporcionaron las bases sobre las cuales descansa este estudio.

7.2 Análisis de la información

Esta etapa podemos decir que fue una de las etapas mas laboriosas, debido a todo el trabajo relativo a la depuración de la información recopilada dentro de la etapa de investigación, ya que se conjuntaron elementos de información, de distinta naturaleza relativos a los siguientes temas: dispersión, dispersión extendible, dispersión dinámica, arboles binarios, arboles dinámicos, arboles B+, arboles virtuales, arboles dinámicos binarios (tries), estructura de la memoria, componentes manejadores de memoria, memoria virtual, funciones de dispersión, Elementos a considerar para el diseño y construcción de funciones de dispersión, Criterios para elegir una función de dispersión perfecta, y algunos tutoriales relativos a la programación orientada a objetos; así como la programación en C++.

A través de esta selección y filtrado de información, se estructuraron los contenidos de los tres primeros capítulos que integran a este documento, Introducción, Antecedentes, y Dispersión Dinámica; lo anterior nos permitió esbozar borrador del contenido del estudio, así como introducir al lector dentro de algunos de los conceptos básicos del área en estudio, y posteriormente dar una visión completa acerca del alcance y magnitud del tema sugerido, para así proporcionar los fundamentos acerca de los elementos necesarios, para llevar a cabo el diseño y construcción de una librería de aplicación.

7.3 Análisis de la aplicación

Una de las etapas mas trascendentales junto con la etapa de construcción del sistema, la constituye la respectiva al análisis del mismo, en nuestro caso este se derivo consecuentemente del objetivo del proyecto, y en relación a su esquema funcional, lo podemos ver conformado por tres elementos fundamentales hacia donde se orientaron los esfuerzos : Función de Transferencia, Memoria Virtual y conformación del Arbol Virtual.

En lo que se refiere principalmente a la ***función de transferencia***, se realizo lo siguiente:

- ✓ Primeramente fue necesaria la búsqueda de elementos de información que nos brindaran aquellos criterios y elementos básicos, necesarios para el desarrollo de la aplicación. Lo anterior debido que a nivel general, para que la aplicación cumpla con el objetivo señalado se hace necesaria la interacción y acoplamiento conjunto de tres elementos: la Función de transferencia que sirve como control de acceso para la administración del almacenamiento de datos los arreglos en la memoria, la estructura que maneja las operaciones de recuperación, inserción y borrado de la información; así como el almacenamiento y disposición de la memoria (Arbol), y por último de las estructuras y las políticas que permiten mantener los datos en la memoria a través de la interacción de elementos de memoria solicitados a los dispositivos de almacenamiento interno.

- ✓ Se manifestó necesaria la dedicación de un tiempo en especial al análisis y diseño de una función de transferencia, lo anterior debido a que se tuvieron que diseñar previamente varias, y se realizaron pruebas con ellas, lo cual apoyo los siguientes resultados: la primera resaltaba un comportamiento estrictamente lineal, y consistía en implementar operaciones algebraicas, para obtener un valor de dispersión asociado, sin embargo al momento de realizar la pruebas con un volumen alto de datos, comenzaba a presentar problemas debido a las colisiones presentadas entre los datos, posteriormente se diseñó otra función, a través de la técnica de división y segmentación.

- ✓ El diseño actual, que se encuentra realizado para la función de transferencia es a través de funciones diseñadas con números no periódicos e irracionales, a través de la inclusión de estos elementos se logro otorgar un mayor grado de confiabilidad a la función de trasferencia, así como también se disminuyo en un porcentaje la posibilidad de las colisiones, ya que las pruebas arrojaron como resultado un optimo direccionamiento de palabras, esta función administra el control de los accesos a la memoria; y fue necesaria adicionalmente

la implementación de un código asociado a cada uno de los caracteres utilizados para la dispersión, lo anterior con el fin de otorgarle un mayor grado de confianza a la función creada.

El diseño y construcción de la función de dispersión, nos permitió establecer mecanismos y parámetros a seguir dentro del desarrollo del sistema, así como nos permitió visualizar las herramientas con las cuales contamos, para el mejoramiento y optimización del desempeño del sistema. Podemos concluir que en el momento en que se cuenta con una función de dispersión optima, de acuerdo a las necesidades de la aplicación principal, podemos decir que se cuenta con un avance practico de mas del 50 %, ya que a través del aseguramiento del funcionamiento del manejo de la memoria, se asegura en un alto porcentaje la integración de los elementos, a través de los cuales se complementara el desarrollo del proceso de operación del sistema.

Con relación a la estructura del ***árbol dinámico*** extendible, podemos considerar lo siguiente como fundamentos concluyentes con base en el análisis realizado:

La estructura del árbol, se encuentra basada en el estudio de los arboles Radix de búsqueda multinivel, (caso especifico de los arboles B+) así como los arboles Binarios de naturaleza dinámica, lo anterior debido a la constante necesidad de expansión y contracción que representan en este caso las funciones de ingreso y baja de registros de información, aparte de ser lo mas parecido dentro de su estructura básica de manejo a un "trie", un árbol binario, dinámico, de naturaleza expandible, no balanceado.

Uno de los aspectos de mayor trascendencia considerado, dentro del análisis es el relativo al tamaño de las paginas de memoria a utilizar, ya que en función de estas ultimas se encuentra la relación de crecimiento de los elementos que apoyaran a la función de dispersión, para un buen control del almacenamiento de información dentro del árbol.

De igual forma se realiza la consideración siguiente: se tiene un elemento extra denominado "pagina critica", dentro de la cual se mantendrán almacenadas las incidencias que necesitan un mayor numero de direcciones y de espacio dentro de la memoria; al menos mas que con la que cuenta la pagina estándar.

Se manifestó necesario de igual forma llevar a cabo un comparativo entre métodos, para tener un estimado del tiempo de ejecución con base en los diferentes tipos de estructuras. Es a través de esta comparación de tiempos como surge el análisis del peor de los casos, para los tiempos de ejecución dentro de la búsqueda de información que se encuentra dentro de la estructura anteriormente mencionada.

Conforme a lo citado anteriormente, en lo que se refiere a la ***memoria virtual*** tenemos lo siguiente:

- ✓ La aplicación del concepto de memoria virtual se realiza a través de la utilización de una cantidad de almacenamiento secundario, la cual se utiliza como mecanismo de almacenamiento adicional para la memoria del equipo, la cual mediante operaciones de intercambio de segmentos y paginas de memoria, permite realizar con mayor agilidad accesos a la información; así como también permite tener el menor numero de accesos al archivo de información.

- ✓ A través de la utilización de la memoria virtual se establecen mecanismos de recuperación de información dentro de la unidad de almacenamiento secundario que permiten desarrollar las tareas de búsqueda, inserción y borrado de elementos, sin necesidad de llevar a cabo mas de un acceso al elemento de almacenamiento físico de la información.

- ✓ Las diferentes técnicas implementadas, para la utilización de la memoria virtual mencionadas dentro del capitulo cuatro, plantean de manera teórica la base del funcionamiento de la que se encuentra implementada dentro del programa Impulso (la aplicación desarrollada para los fines de la tesis), esta se encuentra basada en las técnicas de swapping dentro del disco duro; esto es, a través de la disposición de una sección especifica del dispositivo de almacenamiento para ser utilizado como una extensión de la memoria principal, lo anterior tiene como fin de brindar una mayor capacidad de espacio, para la carga de elementos, y son estos los procesos a través de los cuales se realizará el proceso de carga y descarga a la memoria principal aquellos segmentos de

información que sean necesarios de utilizarse, conforme se avance dentro de la secuencia de llamada dentro de la propia aplicación.

La estructura y manejo de la memoria virtual, se analizaron detenidamente, y debido a que el manejo proporcionado por el Windows 3.1 y 95 cumplía con los esquemas deseados para su administración y control, se decidió utilizar a esta como base, con efecto de optimizar al máximo tiempos en el análisis, diseño y construcción de un proceso que llevara a cabo la administración y control de la memoria, ya que simplemente ello implicaría el desarrollo de una investigación tan profunda y el esfuerzo semejante al del desarrollo de la tesis misma.

7.4 Diseño y desarrollo de la aplicación

Dentro de los aspectos que sobresalen acerca del diseño y desarrollo de la aplicación podemos citar los siguientes comentarios concluyentes :

En lo relativo a esta parte, se comenzó a partir de la selección del lenguaje de programación, ya que este debería de ser un lenguaje que nos permitiera llevar a cabo una programación depurada en cuanto se refiere a la programación orientada a objetos, en nuestro caso en particular elegimos el Borland C++ Versión 4.5, debido a las características propias de manejo del lenguaje, las bondades relativas a la programación de clases que integraran a los objetos, así como contar con las funciones de manejo auxiliar de la memoria dinámica, y un buen entorno de programación en lo que se refiere a la librería OWL.

El cambio de la forma de programación tradicional (procedural), a la nueva filosofía de programación (orientada a objetos), implico llevar a cabo una investigación de conceptos, ideas y metodología , lo cual implico dar una nueva panorámica a la programación desde puntos de vista, tales como declaración de objetos, variables: publicas y privadas; objetos anidados, funciones de herencia, funciones fraternas, etc.

Por otra parte el análisis del algoritmo, nos permitió establecer de primera instancia algunos de los elementos y estructuras de programación, consideradas como necesarias para llevar cabo el desarrollo de procesos involucrados dentro del mismo, objetos, estructuras de almace-

namiento de información, variables, contadores, registros, archivos, manejos de corrientes (archivos) de lectura y escritura, entre otros.

Dentro del diseño especifico de la función de transferencia, se hizo necesario manejar procesos adicionales de conversión de valores decimal a binario, así como también una función denominada espejo(), la cual se encarga de invertir la cadena de caracteres para regresar el valor de conversión.

Asimismo podemos comentar que el mayor esfuerzo se realizo, con respecto a la función y los tipos de operaciones que debíamos elegir, con el fin de poder llevar a cabo la dispersión de los elementos dentro de las direcciones lógicas, trabajamos con cinco funciones de transferencia distintas, una de ellas fue siguiendo el algoritmo de Cichelli, pero sinembargo, se constato que para conjuntos que sobrepasan los 30 elementos resultan completamente infuncionales, de igual forma se comenzó a hacer pruebas con incrementos de desplazamiento en 64 bits, pero se generaban hasta un 45 % de colisiones dentro de la distribución, por lo que se opto por implementar una basada en el truncamiento y división, a través de funciones, otorgando resultados semejantes al anterior.

Con fundamento en lo anterior, se tomo la decisión de establecer una función un tanto mas compleja, a través de la intervención de números irracionales no periódicos, razón por la cual llegamos a la consideración de los números pi y e, y con base en ellos manejamos los valores ingresados a través del código asociado a las letras.

Este código, debido a las pruebas desarrolladas, se decidió crearlo y conformarlo con 78 elementos , los cuales fueron los considerados de uso cotidiano, y en lugar de tomar su valor en código ASCII, generamos nuestra propia tabla de valores, con valores no mayores a 10, debido a que como se procesa carácter por carácter, y se desarrolla una operación de potencia con estos valores; se hace necesario manajar valores, progresos y esto se mantiene a través de su uso, y el manejo de un buen numero de cifras significativas, ya que estos son los elementos de donde obtenemos los valores de la función de dispersión.

De las anteriores aseveraciones podemos concluir, que a través de la definición de una tabla de códigos propia, así como de la construcción de una función de dispersión, un poco más compleja de lo normal, se ha logrado un grado acceder de confiabilidad de la función de transferencia de un 85 %, que dentro de los rangos establecidos, para la funciones que forman mayor parte de ellas y bajo el método empleado, es un grado de optimización de la función de distribución bastante alto, ya que las colisiones presentadas a una sola incidencia son del 14 % y el restante 1 % llegan a tener en un máximo de hasta dos incidencias consecutivas sobre el mismo valor de dispersión.

En lo que se refiere a la estructura del árbol utilizada fue necesario establecer mecanismos de apoyo para el recorrido y búsqueda dentro del árbol, ya que aunque para el almacenamiento y administración se emplean los estándares de la memoria, la recuperación y movimiento de los datos requieren de elementos adicionales para su manejo.

7.5 Pruebas

Es difícil a veces evaluar el trabajo por un punto de desempeño en específico, sin embargo este tipo de aplicación requirió de un proceso de pruebas, que permitiera visualizar específicamente la utilidad de la aplicación, y que adicionara un grado de confiablidad a la misma.

Analizar y tener una visión de los procesos involucrados y de como estos se encuentran vinculados entre sí, es una de las particularidades que brindan la capacidad de visión y perspectiva a una aplicación, el desarrollo de un código o un algoritmo, nos da la indicación de que hacer, más no nos da la capacidad y dinámica de funcionamiento del producto terminado, esto puede ser lo que a muchas innovaciones haga parecer sencillas y elementales, pero es también el detalle que puede hacer que lo sencillo y elemental sea algo practico.

En cuanto a las pruebas desarrolladas resalta lo relativo al índice de dispersión, ya que con fundamento en los resultados podemos concluir que la función de dispersión implementada, puede cubrir un amplio rango de valores, siempre y cuando se cuente con un equipo que brinde la suficiente memoria para establecer indices más altos, debido al consumo de recursos . Es importante resaltar que es bueno, pero aún es optimizable a través del diseño de una función

de dispersión con uso de grafos y funciones no periódicas, que brinden la capacidad de ocupar un mínimo de recursos, aunque valdría la pena hacer una consideración, si la tendencia es la expansión de los recursos y el aprovechamiento de los mismos, ¿por que no?.

Actualmente la capacidad de los discos duros así como su tiempo de acceso, resulta un parámetro fuerte de decisión, ya que esto implicará la utilización o integración de nuevas herramientas de software, para nuestro caso en específico podemos observar que este dispositivo propiamente puede llegar limitante, si carecemos de los recursos de Memoria RAM necesarios, pero contamos con un buen disco duro con tiempos de acceso bajos, puede ayudar a que en el número de accesos necesarios óptimos 1.359 llegue a ser lo suficientemente ràpido y confiable, y que otorgue ventajas en lo relativo al tiempo de procesamiento, aunque no proporcione variaciòn dentro del número de accesos. Lo óptimo sería hacer pruebas con mecanismos de memoria que permitan accesos intermedios que obligarán a realizar sólo una lectura al disco.

De manera general sólo un par de comentarios nos parece necesario realizar:

- ✓ La aplicación resultante de este proyecto se vislumbra como un producto fruto del esfuerzo, tenacidad y resistencia, ya que realizarlo implico pruebas de paciencia, persistencia y hasta un poco de yoga, ya que en los momentos críticos implico el desdoblamiento de ideas, a través de "peloterapia" de las ideas frustadas. Pero independientemente de ello, implico la realización de un anhelo fundamental, terminar con lo que comenzó hace más de cinco años, un compromiso, una meta, un objetivo : La Ingeniería.

- ✓ El objetivo de la Tesis : cumplido, desde todos los puntos de vista, su conformación, su idea, concepción y modelo, en sí este documento integra la aplicación de los conocimientos adquiridos a la largo de la carrera, dentro de las aulas y fuera de ellas, y en lo que se lleva a lo largo de esta vida, en sí no solo representa y pretende cumplir un trámite, sino con una convicción de finalizar con una etapa más de ella.

Capítulo 8

Bibliografía

Libros

Bertizz, A.T. Data Structures Theory and Practice. Computer Science and Applied Mathematics. Academic Press. Inc. New York, New York.1975.

Kruse, Robert L. Estructuras de datos y Diseño de programas. Prentice Hall Hispanoamericana. México. 1989.

Collins, William J. Data Structures and Object Oriented Approach. Addison Wesley Reading, Mass. 1982.

Samet, Hanan. The Design and Analysis of Spatial Data Structures. Addison Wesley Reading, Mass. 1978.

Nievergelt, Jurg and Klaus H. Hinrichs. Algorithms and Data Structures with Applications to Graphics and Geometry. Prentice Hall, New York. 1979.

Larson, P. Dynamic hashing. BIT18; 1978. Pags.184-201.

Aaron M. Tenenbaum; Yedidyab Langsam & Moshe J. A. Data structures using C. Pretince Hall, Englewood Cliffs, N. J.. 1990.

Budd, Timothy A. Classic data structures in C++. Addison-Wesley. Reading, Mass.. 1994.

Bill Zoellick and Michale J. Folk. File Structures. Addison - Wesley Reading, Mass. 2nd. Ed.1992.

Schildt, Herbert. "C++ : Guía de autoenseñanza". McGraw Hill. México. 1995.

Frank B. Brokkenend and Karel Kubat. "C++ Annotations Versión 4.2.4".ICCE, University of Groningen Westerhaven 16, 9718 AW Groningen. Netherlands Published at the University of Groningen. ISBN 90 367 0470 7.1994 - 1997.

Chris H. Papas and William H. Murray. "Manual de Borland C++ 4.0". Mc Graw Hill. España. 1994.

Coad, Peter and Yourdan, Edward. "Object - Oriented Analysis". Secon Edtion. Prentice Hall Building. N.J.. U.S.A.. 1990

Artículos

Sprugnoli, Renzo. Perfect dispersióning Functions : A single Probe Retrieving Method for Static Sets. Communications of the ACM. Vol. 20; No. 11; November 1977. New York, New Jersey. Pags 841-850.

Cichelli, Richard J. Minimal Perfect Hash Functions made simple. Communications of the ACM. Vol 23; No. 12; December 1981. New York, New Jersey. Pags. 17-19.

Jaeschke, G; Oesterburg, G. Technical Correspondence on Cichelli's Minimal Perfect dispersión Functions Made Simple. Communications of the ACM. Vol 23; No. 12 December 1980. New York, New Jersey. Pags 728-729.

Jaeschke, G. Reciprocal dispersióning : A Method for Generating Minimal Perfect dispersióning Functions. Communications of the ACM. Vol 24; No. 12; December 1981. New York, New Jersey. Pags. 829-833.

Severance, Dennis & Duhne Ricardo. Practitioner's Guide to Addresssing Algorithms. Communications of the ACM. Vol. 19; No. 6; June 1976. New York, New Jersey. Pags. 316-324.

Fagin R.; J. Nievergelt; N. Pipeenger & H. R. Strong. extendible hashing - a fast access method for dynamic files-. ACM Transactions on database systems. Vol. 4; No. 3; September 1976. New York, New York. Pags. 315-344.

R. J. Enbody & H. C. Du. Dynamic hashing schemes. Computing Surveys ACM . Vol. 20; No. 2. 1988. New York, New York. Pags. 85-113.

Scholl, M. New file organizations based on dynamic hashing. ACM Transactions on databse systems; Vol. 9. No. 1. March 1981. New york, New York. Pags. 194-211.

Severance, D. & Duhne, R. A practitioner's guide to addresing algorithms.: Communications of the ACM. Vol. 19. No. 6. June 1976. New York, New York. Pags. 314-326.

Gorge Havas and Bohdan S. Majewski. "Optimal algoritms for minimal perfect hashing". Technical report number 234. Key Centre for Software Technology. Departament of computer Science. University of Queensland. Australia. 1992.

Úlfar Erlingsson[a] , Mukkai Krishnamoorthy[a] and T. V. Raman[b]. "Eficient Multiway radix Search Tree". [a]Rensselaer Polytechnic institute, Troy, NY 12180. [b]Advanced Technology Group, Adobe Systems, Mountain View, CA 94039. Septiembre 1996

Internet

C. J. "Keith" van Rijsbergen. Four : File Structures.
http://www.dcs.glasgow.ac.uk/Keith/Chapter.4/Ch.4.html. 1996.

Data structure - PC Webopaedia Definition and Links.
http://www.pcwebopaedia.com/data_structure.html. 1997.

Adminstración de la memoria.
http://info.pue.udlap.mx/udla/clases/so/04.html. 1995.

CIS 307 : Virtual Memory
http://joda.cis.temple.edu/~ingargio/cis307/readings/virtrual.html.1997.

Apéndice A

Tabla de Códigos

Impulso emplea un juego de 78 caracteres diferentes para trabajar. Cada carácter esta representado por dos números. La primer columna es el valor que tienen los caracteres en la tabla de codigos ASCII; la segunda columna es la nueva propuesta de valores que utiliza la aplicación, siendo el mismo juego de caracteres.

ASCII	Código Alterno	Símbolo	ASCII	Código Alterno	Símbolo
32	9.9		78	3.9	N
33	0.1	!	79	4.0	O
34	0.2	"	80	4.1	P
35	0.3	#	81	4.2	Q
36	0.4	$	82	4.3	R
37	0.5	%	83	4.4	S
38	0.6	&	84	4.5	T
39	0.7	'	85	4.6	U
40	0.8	(	86	4.7	V
41	0.9	)	87	4.8	W
42	1.0	*	88	4.9	X
43	1.1	+	89	5.0	Y
44	1.2	,	90	5.1	Z
45	1.3	-	97	5.2	a
46	1.4	.	98	5.3	b
47	1.5	/	99	5.4	c
48	1.6	0	100	5.5	d
49	1.7	1	101	5.6	e
50	1.8	2	102	5.7	f
51	1.9	3	103	5.8	g
52	2.0	4	104	5.9	h
53	2.1	5	105	6.0	i
54	2.2	6	106	6.1	j
55	2.3	7	107	6.2	k
56	2.4	8	108	6.3	l
57	2.5	9	109	6.4	m
65	2.6	A	110	6.5	n
66	2.7	B	111	6.6	o

67	2.8	C	112	6.7	p
68	2.9	D	113	6.8	q
69	3.0	E	114	6.9	r
70	3.1	F	115	7.0	s
71	3.2	G	116	7.1	t
72	3.3	H	117	7.2	u
73	3.4	I	118	7.3	v
74	3.5	J	119	7.4	w
75	3.6	K	120	7.5	x
76	3.7	L	121	7.6	y
77	3.8	M	122	7.7	z

Apéndice B

Impulso ver. 1.0

Manual del usuario

El propósito de la creación de Impulso es el de diseñar y construir un sistema que permita llevar a cabo la implementación de las operaciones y procesos que son parte del algoritmo analizado. Lo anterior, dentro de una aplicación didáctica que permita adicionar mejoras a su desempeño; así como conjuntamente se logren optimizar los recursos de almacenamiento dentro de la memoria, a través de la utilizacion de mecanismos de administración de memoria, todo ello presentado al usuario dentro del marco, de un medio ambiente gráfico, amigable y conocido (Windows).

Los objetivos que sustentan lo anterior, es primeramente contar con un mecanismo de acceso que permita a traves del menor numero de accesos, mantener los elementos de informacion disponibles en cualquier momento, de tal manera que estos puedan ser utilizados durante el proceso de ejecucion; asi como tambien se pretende poder contar con la capacidad de almacenamiento necesaria, a traves del manejo de una estructura dinamica que permita la adición y retiro de elementos, sin tener que comprometer de manera definitiva los recursos de memoria con los cuales cuenta el equipo.

Ambos objetivos pretenden mejorar el desempeño de la aplicación dentro de los ambitos relativos a la carga de la información, y almacenamiento de la misma, así como consecuentemente logren disminiur tiempos dentro de la realizacion de los procesos de busqueda y reemplazo, como en el caso que actualmente se presenta.

Impulso para Windows versión 1.0 es una aplicación que se desenvuelve en el manejo y la edición de archivos de texto. Al momento de empezar a trabajar con el surgirán varias dudas acerca de su funcionamiento y desempeño. Es por eso que se ofrece el siguiente manual, el cual ha sido elaborado con la finalidad de resolver dudas y cuestionamientos sobre el funcionamiento de Impulso.

El manual se dividió en tres partes : la primera, ofrece una breve guía de instalación del programa; la segunda, muestra las características más relevantes del programa a través de su interfaz; y la tercera, una breve descripción de cada una de las opciones que integran el programa. Además, a estas tres partes se le agrega un pequeño compendio sobre los errores que presenta la aplicación.

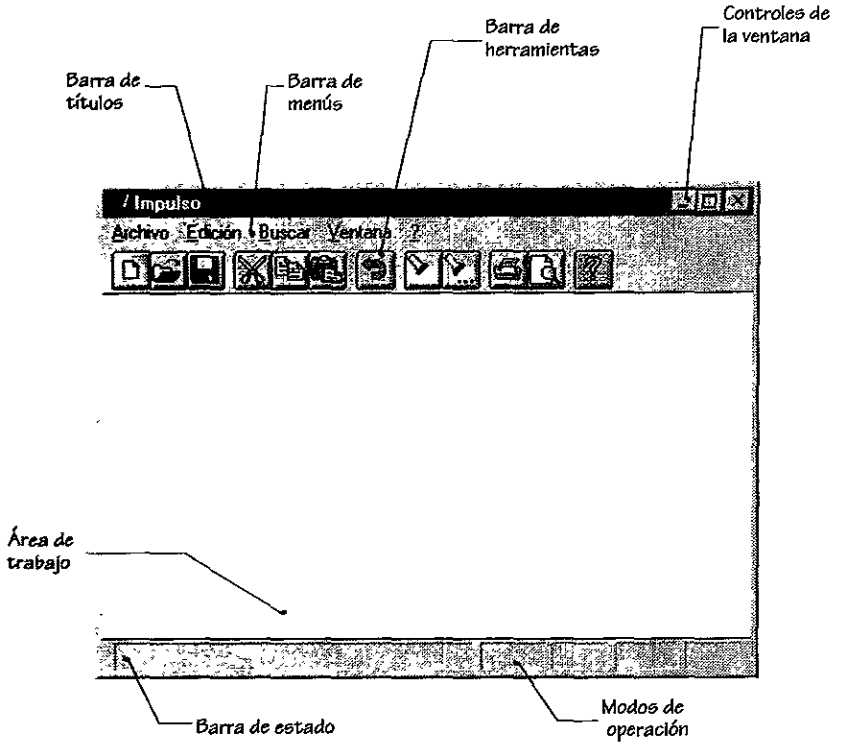
Por último, en el transcurso del manual encontrará unas notas, advertencias y sugerencias sobre el manejo de Impulso. Tomelos en cuenta, pueden ahorrarle algunos dolores de cabeza.

Para instalar Impulso en su computadora, necesita cumplir los siguientes requerimientos :

- ✓ Procesador 486 DX2 a 50 Mhz.
- ✓ Monitor VGA.
- ✓ 2.4 Mbytes de espacio en disco duro.
- ✓ 4 MBytes de memoria RAM.
- ✓ Windows 95 / NT / 3.1x.

Interfaz con el usuario

La presente figura muestra la pantalla principal de Impulso. Cuando se ejecuta, esta es la imagen que le dará la bienvenida a la aplicación :



La ventana principal se compone de varios elementos, los cuales le permitirán trabajar sus documentos de una manera rápida y sencilla. A su vez, Impulso se desempeña bajo el esquema de *multiples documentos / multiples ventanas*. Con esta opción se pueden ver varios documentos a la vez, realizar diversas operaciones de edición con ellos, sin perder de vista los cambios hechos a los documentos.

Los elementos señalados en la ventana principal se describen a continuación :

- ✓ Controles de la ventana

  Estos 3 botones le permiten minimizar, maximizar o cerrar la ventana principal de trabajo.

- ✓ Barra de estados

  Le indicará la acción que realizará el comando o la opción que pretenda utilizar; ya sea a través de la barra de herramientas o por medio de las opciones que integran la barra de menús.

- ✓ Barra de herramientas

  Le permite utilizar una limitada serie de comandos para el manejo y procesamiento de los documentos. Esta barra actúa de acuerdo con el documento activo que se este trabajando.

- ✓ Barra de menús

  En ella se pueden localizar las opciones que se han creado explicitamente para el manejo y procesamiento de los documentos. Así, mismo, se pueden realizar diversas tareas con la ventana principal y las ventana de los documentos que se carguen en ella. Además de accesar la ayuda del programa.

- ✓ Barra de títulos

  En ella se puede localizar el nombre del programa. Y cuando se maximiza la ventana del documento activo, aparece tambien el nombre del documento en cuestión.

- ✓ Modos de operación

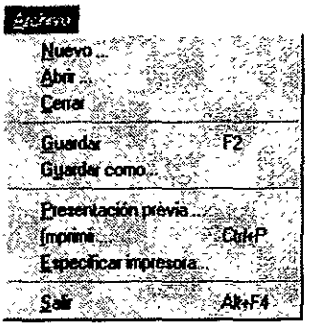
  En este elemento se pueden visualizar los modos de trabajo activos; como tambien las teclas de operación, como son las de activar las letras mayúsculas, activar el tecla númerico o la de bloquear el teclado.

Menús

En ellos se encuentran definidas todas las operaciones que se pueden realizar en Impulso. La barra de menús se encuentra integrada por 5 elementos, los cuales son :

- ✓ Archivo
- ✓ Edición
- ✓ Buscar
- ✓ Ventana
- ✓ ? (Ayuda)

Los cuales serán descritos a continuación :



Este menú es el encargado de realizar el manejo de los documentos. Te permite crear nuevos documentos, cargar los ya creados o bien cerrarlos, siempre cuando esten activos. Asi mismo puedes salvar tu información con el nombre inicial o bien darle uno nuevo.

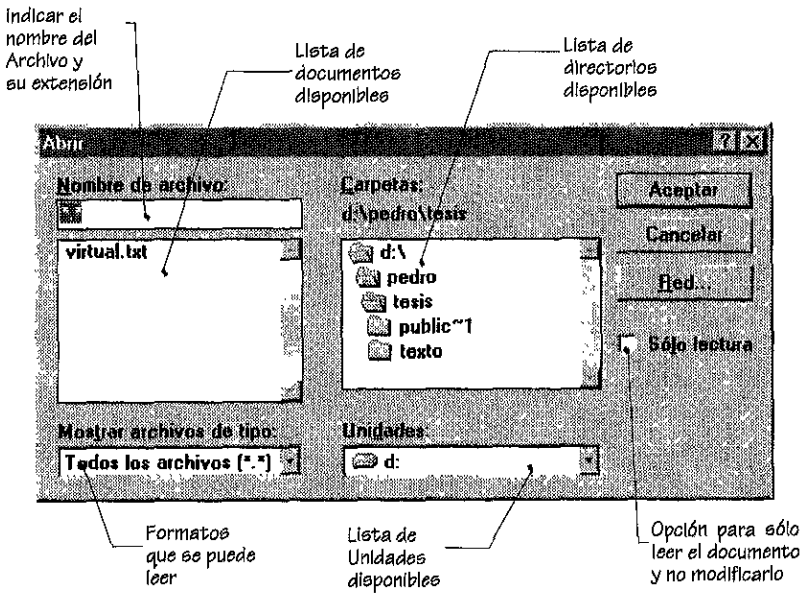
Te permite además tener una vista previa de tu documento. Con el fin de que puedas ver el documento tal y como se vera impreso. Si tienes acceso a otras impresoras, puedes accesar a ellas desde él, configurarlas de acuerdo tus necesiades y definir el tamaño del papel. Si ya terminaste de trabajar con Impulso, este menú te brinda la opción de cerrarlo.

Nuevo...

A través de esta opción puedes crear un nuevo documento dentro de Impulso. Posterior a la utilización del mismo aparecerá la ventana de trabajo correspondiente al nuevo archivo.

Abrir...

Este comando en específico, le permite abrir el(los) documento(s) en disco, y lo(s) coloca dentro de una ventana de trabajo a la que identifica con el nombre del archivo. La lectura y posterior escritura del archivo se puede hacer desde disco flexible o disco duro. En cuanto utilizamos este comando, aparecerá una ventana de dialogo como la siguiente :



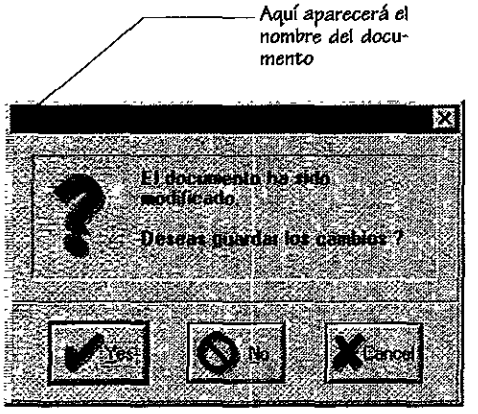
Una vez que aparece la ventana de dialogo, lleve a cabo las siguientes instrucciones :

- ✓ Seleccione la unidad de disco en donde *se encuentra el archivo a recuperar.*
- ✓ Seleccione en su caso el directorio o subdirectorio de trabajo en donde se encuentra el archivo.
- ✓ Seleccione el tipo de archivos a mostrar, con la finalidad de crear la lista para proceder a la identificación del nombre del documento.
- ✓ Identifique y seleccione el archivo deseado.
- ✓ Una vez que este aparezca en el espacio denominado Nombre del archivo, oprima el botón de *Aceptar.*

En caso de tener alguna duda acerca del proceso o bien, se lleve a cabo una selección errónea del mismo, oprima el botón de *Cancelar.*

Cerrar

Este comando le permite cerrar el documento activo. En el caso de que se desee cerrar un archivo el cual ha sido modificado y no ha sido almacenado, *Impulso desplegará la siguiente ventana:*

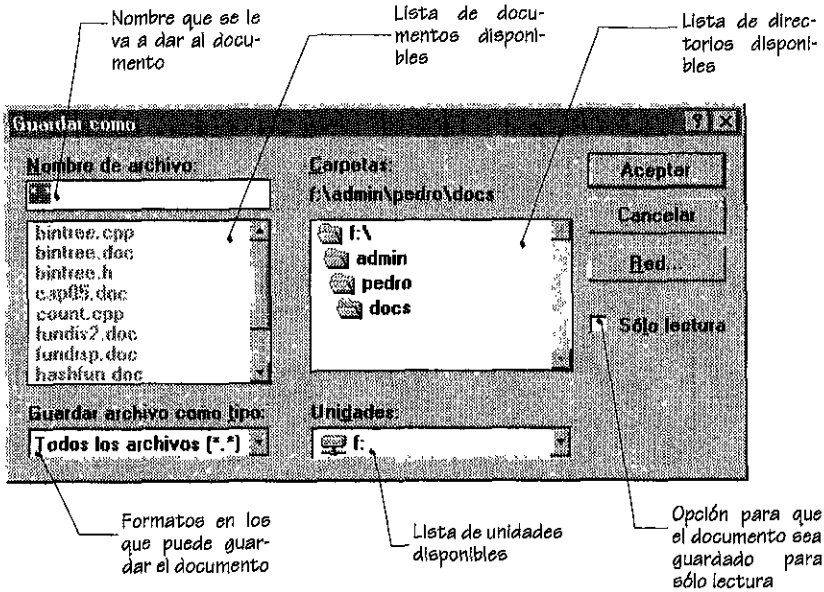


El mensaje de la misma es muy concreto, se deberá seleccionar a conveniencia del usuario; ya que en el caso de que se deseen guardar los cambios, se deberá oprimir el botón con la leyenda *Yes*; en caso contrario, se deberá oprimir *No*; y la opción de *Cancel*, se deberá utilizar solamente para evitar que se lleve a cabo cualquiera de las otras dos opciones y cerrar esta ventana.

Guardar F2

Esta opción le permite almacenar el documento que se encuentre abierto en la ventana activa en un archivo.

En cuanto accionamos dicho comando por primera vez, es decir, es nuevo el documento y no se ha guardado anteriormente, aparecerá una ventana de dialogo como la siguiente :



Una vez que aparece la ventana de dialogo, se sugieren las siguientes instrucciones :

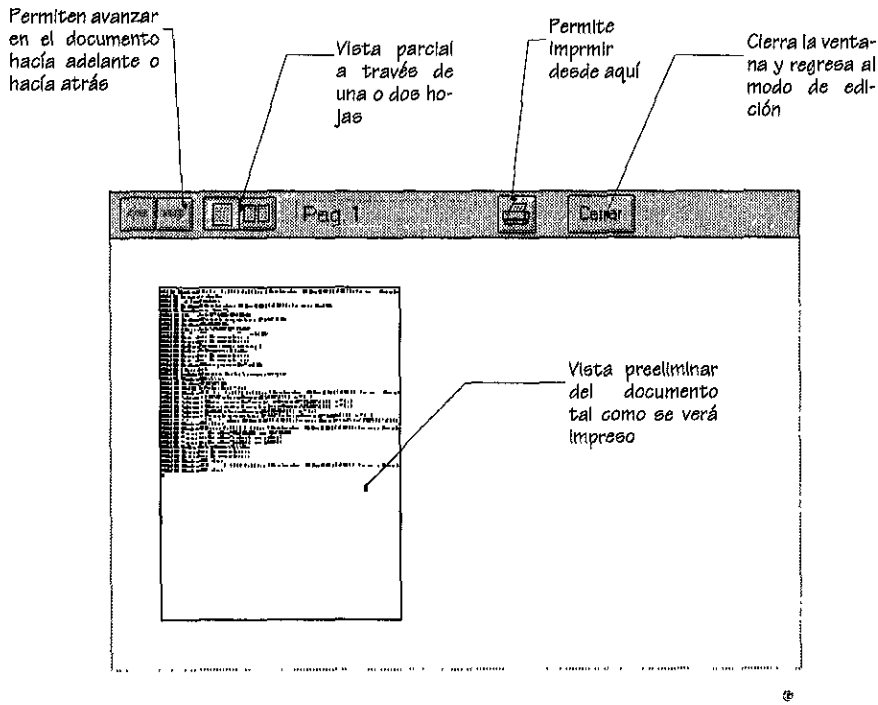
- ✓ Seleccione la unidad de disco en donde desea almacenar su información.
- ✓ Posteriormente, seleccione el directorio o subdirectorio de trabajo en donde habrá de guardarse.
- ✓ Seleccione de igual forma el formato, en el cuál desea almacenar el documento.
- ✓ En el espacio correspondiente al Nombre del archivo, teclee el nombre con el cuál identificará al documento.
- ✓ Presione el botón de *Aceptar*.

Una vez que se presiona *Aceptar*, el sistema procede a realizar instantáneamente la operación, y se encuentra en espera de la siguiente instrucción a desempeñar.

Para el caso de que no sea la primera vez que se manda guardar, el comando se ejecuta de manera automática, respetandose el nombre anterior, así como la unidad de disco y el directorio de trabajo donde se encuentra ubicada la última copia almacenada.

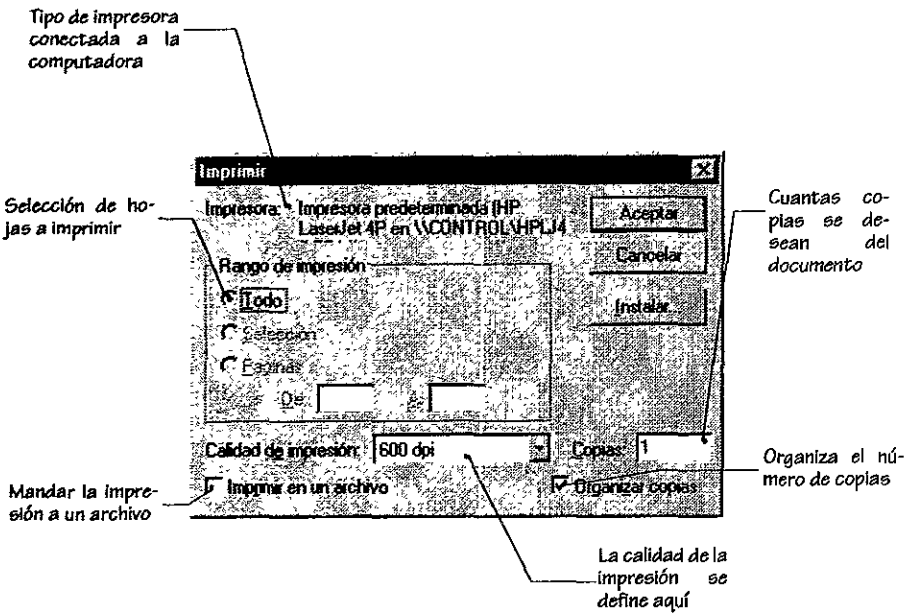
Presentación previa ...

Ofrece una vista preeliminar del documento, tal y como se ve en la siguiente figura :



Imprimir Ctrl + P

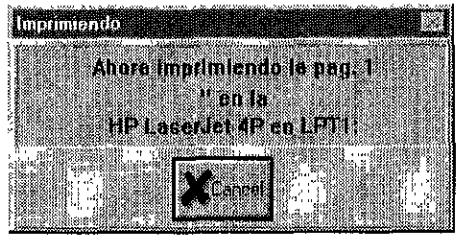
Puede(s) imprimir tu(s) documento(s) utilizando esta opción. Al momento de utilizar este comando, aparecerá la siguiente ventana de dialogo :



Una vez que aparece la ventana de dialogo, a continuación se ofrecen algunas instrucciones que le pueden ayudar en el manejo de esta ventana :

- ✓ Seleccione el rango de impresión que desee, con respecto al documento cargado dentro de la ventana activa.
- ✓ Posteriormente seleccione la calidad de impresión requerida para el documento.
- ✓ Seleccione de igual forma, si desea almacenar la impresión dentro de un archivo.
- ✓ En el espacio correspondiente al Número de copias, teclee el número de las copias que son requeridas.
- ✓ Presione el botón de *Aceptar*.

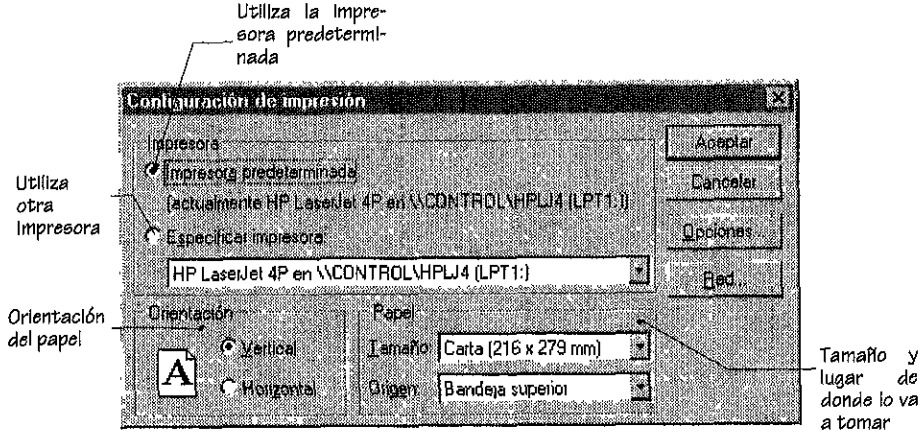
Una vez aceptada la operación, aparecerá la siguiente ventana de dialogo :



La cuál le informa el estado actual de la impresora, la página que se esta imprimiendo en ese instante y a que impresora se esta enviando el documento. Además, ofrece la opción de poder cancelar la impresión en el momento que lo desee.

Especificar impresora ...

Aporta los elementos necesarios para seleccionar el modelo y las particularidades acerca del dispositivo de impresión y la presición de algunas características del documento a imprimir. En el momento de accionar este comando, aparecerá una ventana de dialogo como la siguiente :



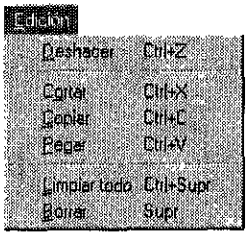
Una vez presente la ventana de dialogo, lleve a continuación estas instrucciones :

- ✓ Seleccione el modelo de la impresora que desee para llevar a cabo la impresión de sus trabajos.
- ✓ Posteriormente seleccione la orientación del papel requerida para llevar a cabo la impresión.
- ✓ Seleccione de igual forma el tamaño del papel en el cuál se va a imprimir, así como el origen de este dentro de la impresora, es decir, de donde se va a proveer el papel para la impresión.
- ✓ En el caso de asi requerirlo, seleccione las opciones de impresora y modificar aquellos parametros que le sean necesarios o que no sean requeridos para la correcta ejecución de la impresión dentro del dispositivo.
- ✓ Presione el botón de *Aceptar*.

Una vez hecho lo anterior, se esta en condiciones de probarlos y ver sí cumplen con el cometido. De no ser así, vuelva a utilizar esta opción y vuelva a hacer los cambios necesarios.

Salir Alt + F4

Esta opción le permite cerrar Impulso. Si tiene varias ventanas abiertas, estas se cerraran de acuerdo con la última acción de guardar, es decir, si modificó alguno de ellos y no guardo estos cambios, Impulso le preguntara si desea guardar estos cambios o bien procede a cerrar el documento sin los mismos.



Dentro de este menú en particular se tienen las herramientas necesarias para llevar a cabo una serie de operaciones en cuestión de la edición de los documentos, como lo es el de copiar selecciones de texto, borrarlas o pegarlas ya sea dentro del mismo documento o a través de varios documentos. También ofrece la posibilidad de limpiar todo el documento o de borrar un caracter o una selección de los mismos.

A continuación se describen a detalle cada uno de los comandos que integran este menú :


Deshacer Ctrl + Z

Si activaste un comando por error o la operación que llevaste a cabo no fue del todo satisfactoria. Esta opción, como su nombre lo indica, deshace la última acción llevada a cabo y regresa el documento a la forma que se encontraba hasta antes de realizar la tarea que fue ejecutada.


Cortar Ctrl + X

Toma una selección de texto realizada dentro de la ventana activa del documento y la remueve del documento, manteniendola dentro del Portapapeles.

 Copiar Ctrl + C

Toma una selección de texto hecha dentro de la ventana activa del documento, y efectúa una copia del mismo. La cuál es depositada en el Portapapeles para su posterior uso.

 Pegar Ctrl + V

Recurre al Portapapeles, enlazando el contenido del mismo. El cuál fue depositado ahí, ya sea por la acción de cortar o copiar texto, y procede a insertar el mismo en el lugar previamente especificado.

Limpiar todo Ctrl + Supr

Elimina todo el texto que se encuentre dentro de la ventana activa del documento.

Borrar Supr

Elimina una selección de texto o caracter por caracter dentro de la ventana activa del documento.

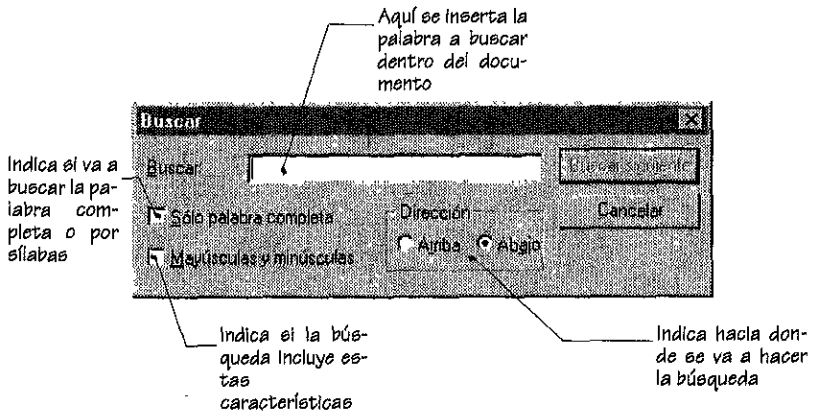
En este menú se encuentran aquellos comandos que son utilizados con la finalidad de revisar incidencias muy especificas dentro del documento. Así como llevar a cabo las operaciones de reemplazo que *pudieran ser convenientes.*

A continuación se describe cada una de las opciones que integran este menú :

Buscar ...

Localiza dentro de la ventana activa la incidencia de una palabra o una selección de texto a través de ciertas características predefinidas por el usuario.

Al iniciar la ejecución del comando, aparecerá una ventana de dialogo como la siguiente :



Una vez que se tiene la ventana de dialogo correspondiente, lleve a cabo las siguientes instrucciones :

- ✓ Seleccione o teclee el texto o palabra a buscar.
- ✓ Posteriormente seleccione las características que desee involucrar en la búsqueda. En este caso que la búsqueda se haga a través de la palabra completa o se haga estrictamente como esta escrita en mayúsculas y minúsculas.
- ✓ Seleccione de igual forma la dirección a través de la cual se realizará la búsqueda dentro del documento.
- ✓ Presione el botón *Buscar siguiente*, acto seguido empezará la búsqueda mos-

trando las incidencias. En caso de no haber, aparecerá el mensaje correspondiente indicandole que no hay más ocurrencias para la palabra en cuestión.

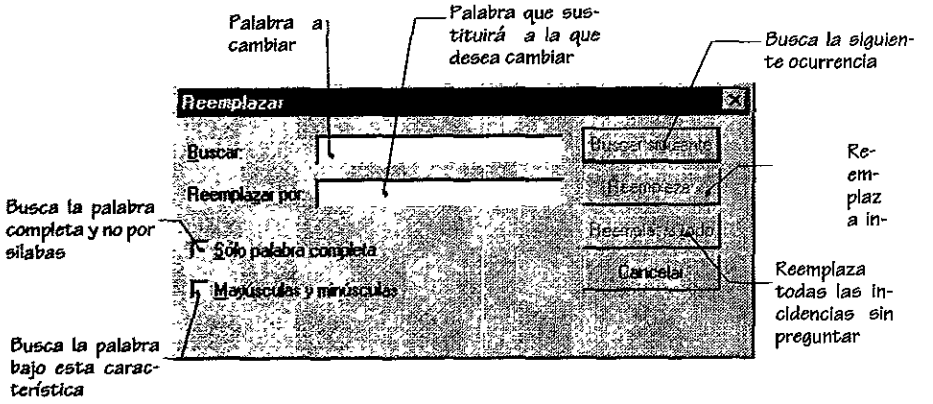
✓ En caso de ser necesario, presione el botón antes citado cuantas veces sea necesario para localizar más incidencias dentro del documento.

Para abandonar está opción, oprima el boton *Cancelar*.

Reemplazar ...

Localiza y cambia dentro de la ventana activa del documento las incidencias de una palabra o de una selección de texto, usando parámetros definidos por el usuario.

Al utilizar este comando, aparecerá una ventana de dialogo como la siguiente :



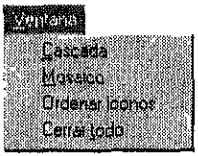
Una vez que se tiene la ventana de dialogo, lleve a cabo estas instrucciones :

✓ Seleccione o teclee la palabra a buscar.

- ✓ Teclee el texto o palabra por la cual se va a reemplazar la anterior.
- ✓ Posteriormente seleccione las características con las cuales quiere que se lleve a cabo el reemplazo.
- ✓ Presione el boton *Buscar siguiente* y dentro de la ventana activa del documento se comenzará a seleccionar las ocurrencias que se vayan presentando.
- ✓ Si la incidencia se requiere cambiar, presione el botón *Reemplazar*.
- ✓ O si se requiere cambiar todas las incidencias que se encuentran en el documento de un sólo golpe , presione el botón *Reemplazar todo*.
- ✓ En caso de ser necesario, presione el botón *Buscar siguiente* cuantas veces se requiera para localizar más incidencias dentro del ventana activa.

Para abandonar esta opción, oprima el botón *Cancelar*.

Realiza la búsqueda de la siguiente ocurrencia dentro del texto, siempre y cuando se mantenga la palabra que inicio la búsqueda anterior. Dicha búsqueda se realiza dentro de cualquier ventana activa de documento.



A través de este menú se pueden llevar a cabo las operaciones que manejan el despliegue y distribución de las ventanas activas abiertas dentro del sistema, así como el control de cierre de las mismas.

A continuación se describe cada una de las operaciones que se pueden realizar a través de la selección de las siguientes opciones :

Cascada

Organiza las ventanas activas en forma de cascada, según la secuencia de apertura de cada una de las ventanas activas.

Mosaico

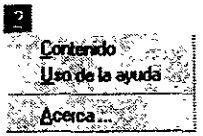
Organiza las ventanas activas de forma equitativa dentro del área de trabajo de Impulso.

Ordenar iconos

Ordena las ventanas activas minimizadas en la parte inferior izquierda del área de trabajo, esto se hace según el orden de apertura de los documentos.

Cerrar todo

Cierra todas las ventanas que se encuentren abiertas en el área de trabajo.



En este menú se tienen las opciones que accesan la ayuda del sistema como de Impulso, además le permite accesar a la ventana de créditos del programa.

A continuación se describen las opciones que integran a este menú :

Contenido

Despliega la ayuda en forma que se pueden localizar los aspectos más generales de Impulso, ya sea por tema, indice general y búsqueda de palabras.

Uso de la ayuda

Brinda las recomendaciones generales que proporciona Impulso acerca del correcto uso de la ayuda, con la finalidad de aprovecharla al máximo.

Acerca ...

Despliega la ventana de creditos , en la cuál se encuentra el nombre de las personas que elaborarón Impulso, la fecha de elaboración y demás características propias de esta opción.

Erratas

Impulso en su primera versión presenta los siguiente errores. Los cuales de alguna manera no alteran su desempeño y funcionamiento.

Impresión

En lo que respecta a la impresión del documento tenemos los siguientes errores :

- ✓ Como habrá podido notar en la pag. C.12. La ventana de dialogo en su sección Rango de impresión, la única opción habilitada es la de Todo, así que cuando mande un documento, este se enviará todo por completo.

- ✓ Mientras que en la página C.13, en la figura correspondiente al estado de la impresión del documento, no aparece el nombre del archivo que se está imprimiendo. Este pequeño error no altera de alguna forma la impresión de su documento.

Edición

En la edición de los documentos tenemos los siguientes :

- ✓ Impulso cuenta con el manejo de una sola fuente, la Times New Roman. Cualquier otro tipo de fuente no podra ser desplegada en pantalla.

- ✓ El manejo del boton derecho con las opciones del menú de Edición funciona bien en Windows 95/NT, mientras que en Windows 3.1x no se encuentra habilitado.

Archivos

En lo referente a los archivos, se tiene la siguiente limitante de operación :

- ✓ En lo relativo a la carga y almacenamiento de los archivos, la aplicación solamente trabaja con las siguientes extensiones :

  - ✓ Documentos de texto (*.txt)
  - ✓ Documentos de texto - formato MS-DOS

- ✓ Dentro del desempeño de Impulso, el uso de diversas ventanas de edición se encuentra en función de la memoria disponible del equipo; estableciendose la siguiente relación :

  Con 8 Mbytes de RAM y abriendo archivos de un tamaño promedio de 32 Kbytes, las ventanas abiertas, sin que la aplicación genere conflictos, serán de 128 (No se considera el uso alternativo de ninguna otra aplicación).

Apéndice C

Código fuente

A continuación el codigo fuente de la aplicación.

Archivo fuente principal IMPLSAPP.CPP

```
#include <owl\owlpch.h>
#pragma hdrstop

#include <dir.h>

#include <dir.h>
#include "implsapp.h"
#include "implmdic.h"
#include "implmdi1.h"
#include "dec2bin.h"
#include "bintree.h"
#include "implsabd.h"                    // Definicion de la caja de dialogo Acerca de.

// Generación del Archivo de Ayuda.
const char HelpFileName[] = "Impulso.hlp";

// Soporte de la función de Tomar / Dejar.
TFileDrop::TFileDrop (char* fileName, TPoint& p, BOOL inClient, TModule* module) {
  char   exePath[MAXPATH];

  exePath[0] = 0;
  FileName = strcpy(new char[strlen(fileName) + 1], fileName);
  Point = p;
  InClientArea = inClient;

  Icon = (WORD)FindExecutable(FileName, ".\\",
                              exePath) <= 32 ? 0 : ::ExtractIcon(*module, exePath, 0);

  // Usa un signo de interrogación si no puede obtener el Icono correspondiente al ejecutable.
  if ((WORD)Icon <= 1) {             // 0=No existen iconos en el exe,  1=not an exe
    Icon = LoadIcon(0, (WORD)Icon == 1 ? IDI_APPLICATION : IDI_QUESTION);
    DefIcon = TRUE;
  } else
    DefIcon = FALSE;
}


TFileDrop::~TFileDrop () {
  delete FileName;
  if (!DefIcon)
    FreeResource(Icon);
}

const char *TFileDrop::WhoAmI () { return FileName; }

//{{ImpulsoApp - Implementación del Algoritmo}}
//{{DOC_VIEW}}
DEFINE_DOC_TEMPLATE_CLASS(TFileDocument, TEditView, DocType1);
//{{DOC_VIEW_END}}

//{{DOC_MANAGER}}
DocType1 __dvt1("Todos los archivos (*.*)", "*.*", 0, "TXT", dtAutoDelete | dtUpdateDir);
```

```
//{{DOC_MANAGER_END}}

//Construye una tabla de respuesta para todos los mensajes/comandos
// manejados por la aplicación.

DEFINE_RESPONSE_TABLE1(ImpulsoApp, TApplication)
//{{ImpulsoAppRSP_TBL_BEGIN}}
  EV_OWLVIEW(dnCreate, EvNewView),
  EV_OWLVIEW(dnClose, EvCloseView),
  EV_COMMAND(CM_HELPABOUT, CmHelpAbout),
  EV_COMMAND(CM_HELPCONTENTS, CmHelpContents),
  EV_COMMAND(CM_HELPUSING, CmHelpUsing),
  EV_WM_DROPFILES,
  EV_WM_WININICHANGE,
//{{ImpulsoAppRSP_TBL_END}}
END_RESPONSE_TABLE;

// Clase - ImpulsoApp
ImpulsoApp::ImpulsoApp () : TApplication("Impulso") {
  HelpState = FALSE;
  ContextHelp = FALSE;
  HelpCursor = 0;

  Printer = 0;
  Printing = FALSE;

  DocManager = new TDocManager(dmMDI | dmMenu);
}

ImpulsoApp::~ImpulsoApp () {
  if (Printer)
    delete Printer;
}

BOOL ImpulsoApp::CanClose () {
  BOOL result = TApplication::CanClose();
  // Cierra el mecanismo de ayuda si se encuentra en uso.
  if (result && HelpState)
    MainWindow->WinHelp(HelpFileName, HELP_QUIT, 0L);
  return result;
}

void ImpulsoApp::SetupSpeedBar (TDecoratedMDIFrame *frame) {
  // Crea la barra de herramientas y  asocia los botones con los comandos.
  TControlBar* cb = new TControlBar(frame);
  cb->Insert(*new TButtonGadget(CM_MDIFILENEW, CM_MDIFILENEW));
  cb->Insert(*new TButtonGadget(CM_MDIFILEOPEN, CM_MDIFILEOPEN));
  cb->Insert(*new TButtonGadget(CM_FILESAVE, CM_FILESAVE));
  cb->Insert(*new TSeparatorGadget(6));
  cb->Insert(*new TButtonGadget(CM_EDITCUT, CM_EDITCUT));
  cb->Insert(*new TButtonGadget(CM_EDITCOPY, CM_EDITCOPY));
  cb->Insert(*new TButtonGadget(CM_EDITPASTE, CM_EDITPASTE));
  cb->Insert(*new TSeparatorGadget(6));
  cb->Insert(*new TButtonGadget(CM_EDITUNDO, CM_EDITUNDO));
  cb->Insert(*new TSeparatorGadget(6));
  cb->Insert(*new TButtonGadget(CM_EDITFIND, CM_EDITFIND));
  cb->Insert(*new TButtonGadget(CM_EDITFINDNEXT, CM_EDITFINDNEXT));
```

```
    cb->Insert(*new TSeparatorGadget(6));
    cb->Insert(*new TButtonGadget(CM_FILEPRINT, CM_FILEPRINT));
    cb->Insert(*new TButtonGadget(CM_FILEPRINTPREVIEW, CM_FILEPRINTPREVIEW));
    cb->Insert(*new TSeparatorGadget(6));
    cb->Insert(*new TButtonGadget(CM_HELPCONTENTS, CM_HELPCONTENTS));

    // Adiciona contextos de ayuda volatiles en línea.
    cb->SetHintMode(TGadgetWindow::EnterHints);

    frame->Insert(*cb, TDecoratedFrame::Top);
}

// Inicialización de la Aplicación.
void ImpulsoApp::InitMainWindow () {
    TDecoratedMDIFrame* frame = new TDecoratedMDIFrame(Name, MDI_MENU,
                                                       *(new ImpulsoMDIClient), TRUE);

    nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL : nCmdShow;

    // Asigna el ICONO con esta aplicación.
    frame->SetIcon(this, IDI_MDIAPPLICATION);

    // Menú asociado con la ventana y la tabla aceleradora asociada a la tabla.
    frame->AssignMenu(MDI_MENU);

    // Asociación con la tabla acelerador.
    frame->Attr.AccelTable = MDI_MENU;

    SetupSpeedBar(frame);

    TStatusBar *sb = new TStatusBar(frame, TGadget::Recessed,
                                    TStatusBar::CapsLock      |
                                    TStatusBar::NumLock       |
                                    TStatusBar::ScrollLock    |
                                    TStatusBar::Overtype);
    frame->Insert(*sb, TDecoratedFrame::Bottom);
    MainWindow = frame;

    // Habilitación de los BWCC (Controles personalizados de Borland para Windows )
    EnableBWCC();
}

// Manejadores de respuesta de la tabla
void ImpulsoApp::EvNewView (TView& view) {
    TMDIClient *mdiClient = TYPESAFE_DOWNCAST(MainWindow-GetClientWindow(), TMDIClient);
    if (mdiClient) {
        ImpulsoMDIChild* child = new ImpulsoMDIChild(*mdiClient, 0, view.GetWindow());

        // Asocia a este ICONO con su ventana hija.
        child->SetIcon(this, IDI_DOC);

        child->Create();
    }
}
void ImpulsoApp::EvCloseView (Tview&) { }

// Ayuda del menu de comandos del contenido
```

```
void ImpulsoApp::CmHelpContents () {
  // Show the help table of contents.
  HelpState = MainWindow->WinHelp(HelpFileName, HELP_CONTENTS, 0L);
}

// Menú de ayuda desplegando los comandos de ayuda.
void ImpulsoApp::CmHelpUsing () {
  // Despliega los contenidos del archivo de ayuda de windows.
  HelpState = MainWindow->WinHelp(HelpFileName, HELP_HELPONHELP, 0L);
}

// Menú de ayuda acerca de los comandos de la aplicación impulso.exe
void ImpulsoApp::CmHelpAbout () {
  // Muestra el dialogo de la ventana modal.
  ImpulsoAboutDlg(MainWindow).Execute();
}

void ImpulsoApp::InitInstance () {
  TApplication::InitInstance();

  // Acepta archivos via tomar/dejar dentro de la ventana de despliegue en uso.
  MainWindow->DragAcceptFiles(TRUE);
}

void ImpulsoApp::EvDropFiles (TDropInfo drop) {
  // Número de archivos dejados.
  int totalNumberOfFiles = drop.DragQueryFileCount();

  TFileList* files = new TFileList;
  for (int i = 0; i totalNumberOfFiles; i++) {
    // Comenta al Tell DragQueryFile el archivo interesado en (i) y la longitud de su buffer.
    int    fileLength = drop.DragQueryFileNameLen(i) + 1;
    char   *fileName = new char[fileLength];

    drop.DragQueryFile(i, fileName, fileLength);

    // Obteniendo el archivo dejado en el proceso. La locación es relativa a las coordenadas del cliente
    // y se tendrán valores negativos si se dejan  en partes que no pertenecen a la ventana del cliente.
    //
    // DragQueryPoint copia este punto donde el archivo fué dejado y retorna entonces si el punto
    // se encuentra o no dentro del área del cliente. Independientemente de que el archivose deje
    // o no en un área de no cliente de la ventana, se seguira recibiendo el nombre del archivo.
    TPoint  point;
    BOOL   inClientArea = drop.DragQueryPoint(point);
    files->Add(new TFileDrop(fileName, point, inClientArea, this));
  }

  // Abre los archivos que fuerón traspasados.
  AddFiles(files);

  // Libera la memoria alojada por este manejador con el DragFinish.
  drop.DragFinish();
}
void ImpulsoApp::AddFiles (TFileList* files) {
  // Abre todos los archivos que fuerón dejados dentro.
  TFileListIter fileIter(*files);
```

```
  while (fileIter) {
    TDocTemplate* tpl = GetDocManager()->MatchTemplate(fileIter.Current()->WhoAmI());
    if (tpl)
      tpl->CreateDoc(fileIter.Current()->WhoAmI());
    fileIter++;
  }
}

BOOL ImpulsoApp::ProcessAppMsg (MSG& msg) {
  if (msg.message == WM_COMMAND) {
    if (ContextHelp || (GetKeyState(VK_F1) 0)) {
      ContextHelp = FALSE;
      MainWindow->WinHelp(HelpFileName, HELP_CONTEXT, msg.wParam);
      return TRUE;
    }
  } else
    switch (msg.message) {
    case WM_KEYDOWN:
      if (msg.wParam == VK_F1) {
        // Si se presionan las teclas Shift/F1 se inicia el conjunto de ayudas del
        // cursor y se enciende el estado de ayuda modal.
        if (::GetKeyState(VK_SHIFT) < 0) {
          ContextHelp = TRUE;
          HelpCursor = ::LoadCursor(MainWindow->GetModule()->GetInstance(),
                                    MAKEINTRESOURCE(IDC_HELPCURSOR));
          ::SetCursor(HelpCursor);
          return TRUE;                    // Despliega el mensaje.
        } else {
          // Si F1 y/o la tecla Shift se presionan invocan al menú principal de la ayuda.
          MainWindow->WinHelp(HelpFileName, HELP_INDEX, 0L);
          return TRUE;                    // Despliega el mensaje.
        }
      } else {
        if (ContextHelp && (msg.wParam == VK_ESCAPE)) {
          if (HelpCursor)
            ::DestroyCursor(HelpCursor);
          ContextHelp = FALSE;
          HelpCursor = 0;
          MainWindow->SetCursor(0, IDC_ARROW);
          return TRUE;                    // Despliega el mensaje.
        }
      }
      break;

    case WM_MOUSEMOVE:
    case WM_NCMOUSEMOVE:
      if (ContextHelp) {
        ::SetCursor(HelpCursor);
        return TRUE;                      // Despliega el mensaje.
      }
      break;

    case WM_INITMENU:
      if (ContextHelp) {
        ::SetCursor(HelpCursor);
        return TRUE;                      // Despliega el mensaje.
      }
```

```
        break;

      case WM_ENTERIDLE:
        if (msg.wParam == MSGF_MENU)
          if (GetKeyState(VK_F1) 0) {
          ContextHelp = TRUE;
          MainWindow->PostMessage(WM_KEYDOWN, VK_RETURN, 0L);
          return TRUE;                          // Despliega el mensaje.

        }
        break;

      default:
        ;
      }; // End of switch

    // Continua con el procesamiento normal
    return TApplication::ProcessAppMsg(msg);
}

void ImpulsoApp::EvWinIniChange (char far* section) {
  if (lstrcmp(section, "windows") == 0) {
    // Si se encuentra el dispositivo cargado dentro de archivo WIN.INI, entonces la impresora
    // puede haber cambiado Si tenemos al manejador TPrinter (Printer) entonces nos
    // cercioremonos y aseguremonos que es identico al dispositivo seleccionado
    // en trando en el archivo WIN.INI.
    if (Printer) {
      char printDBuffer[255];
      LPSTR printDevice = printDBuffer;
      LPSTR devName = 0;
      LPSTR driverName = 0;
      LPSTR outputName = 0;

      if (::GetProfileString("windows", "device", "", printDevice, sizeof(printDevice))) {
        // La cadena que debe de regresar es algo como esto:
        //     HP LaserJet III,hppcl5a,LPT1:
        // Donde el formato es :
        //     devName,driverName,outputName
        //      Nomdisp,Nomanj, Nomsal

        devName = printDevice;
        while (*printDevice) {
          if (*printDevice == ',') {
            *printDevice++ = 0;
            if (!driverName)
              driverName = printDevice;
            else
              outputName = printDevice;
          } else
            printDevice = AnsiNext(printDevice);
        }
        if ((Printer->GetSetup().Error != 0)                                    ||
          (lstrcmp(devName, Printer->GetSetup().GetDeviceName()) != 0)   ||
          (lstrcmp(driverName, Printer-Z>GetSetup().GetDriverName()) != 0) ||
          (lstrcmp(outputName, Printer->GetSetup().GetOutputName()) != 0)) {

          // Impresora nueva instalada, obtiene el nuevo dipositivo de impresión ahora.
```

```
            delete Printer;
            Printer = new TPrinter;
          }
        } else {
          delete Printer;                    // Ninguna impresora instalada (GetProfileString fallido).
          Printer = new TPrinter;
        }
      }
    }
  }

int OwlMain (int , char* []) {
  ImpulsoApp    App;
  int           result;

  result = App.Run();
  return result;
}
```

Archivo *cabecera* DEC2BIN.H

```
# include <conio.h>
# include <iostream.h>
# include <string.h>
# include <stdlib.h>
# include <math.h>

// Clase que se encarga de realizar la conversion
// de numero decimal a uno binario.

class dec2bin {
        int  indice;
        long numero;
        char *string;

        void ReasignaMem(void);
        void Conversion(void);

 public :
        dec2bin();
        dec2bin(long valor);
        ~dec2bin() { delete [] string; }

        void AsignaNum(long valor);
        void Espejo(void);
        char const *NumeroBin(void) const;
        unsigned const NumeroBits(void) const;
};


// Clase que se encarga de convertir una cadena de
// caracteres a un valor numerico.
// Funcion de Dispersion H(Key).

class Dispersion : public dec2bin {
        float  Valor;
```

```
        int   lstring;
        float  Decimal;
        char   *String;

        void   Proceso();
        double Suma_ASCII();
        double NvCodigo(int Ctr);
        double  ObtenNm(double Ptr);
        void Calcula();

 public :
        Dispersion();
        Dispersion(int Ndecimal,char *Tstring);
        ~Dispersion () { delete String; } ;

        void AsignaV(int Ndecimal,char *Tstring);
        long const NumeroObt() const;
        char const *NumeroFn() const;
};
```

Archivo fuente DEC2BIN.CPP

```
# include "dec2bin.h"

// Funciones miembro de la clase dec2bin.

dec2bin::dec2bin() {
 indice = 1;
 numero = 0;
 string = new char [ indice ];
 string[0] = '\0';
}

dec2bin::dec2bin(long valor) {
 indice=1;
 if(0 <= valor && valor < 65536L)
        numero = valor;
 else {
        cout << " Numero a convertir no valido. " << endl;
        exit(1);
 }
 string = new char [ indice ];
 string[0] = '\0';
 Conversion();
}

void dec2bin::AsignaNum(long valor) {
 if(0 <= valor && valor < 65536L )
        numero = valor;
 else {
        cout << " Numero a convertir no valido. " << endl;
        exit(1);
 }
 indice = 1;
 delete [] string;
 string = new char [ indice ];
```

```
 string [0] = '\0';
 Conversion();
}

void dec2bin::Conversion() {
 if(!numero || numero==1) {
        if(!numero)
          strcat(string,"0");
        else
          strcat(string,"1");
        Espejo();
        return;
 }
 else {
        if(!(numero % 2))
          strcat(string,"0");
        else
          strcat(string,"1");

        numero /=2, indice++;
        ReasignaMem();
        Conversion();
 }
}

char const *dec2bin::NumeroBin() const { return string; }

unsigned const dec2bin::NumeroBits() const { return unsigned(indice); }

void dec2bin::Espejo() {
 char *temp = new char [indice];
 int j = 0; temp[j] = '\0';

 for(int i = indice-1; i > -1; i--)
   (char) temp[j] = (char) string[i], j++;

 strcpy(string,temp);
 string[indice] = '\0';
 delete [] temp;
}

void dec2bin::ReasignaMem() {
 char *stemp = new char [indice];
 strcpy(stemp,string);
 delete [] string;
 string = new char [indice];
 stemp[indice-1] = '\0';
 strcpy(string,stemp);
 string[indice] = '\0';
 delete [] stemp;
}

// Funciones miembro de la clase  Dispersion.

Dispersion::Dispersion() {
 Valor = Decimal = 0.0;
 lstring = 0;
```

```
 String = '\0';
}

Dispersion::Dispersion(int Ndecimal,char *Tstring) {
 Valor = 0.0;
 lstring = strlen(Tstring);
 String = strdup(Tstring);
 if(2 <= Ndecimal && Ndecimal <= 5)
   Decimal = pow(10,Ndecimal);
 else
   Decimal = pow(10,2);
 Calcula();
}

void Dispersion::AsignaV(int Ndecimal,char *Tstring) {
 Valor = 0.0;
 lstring = strlen(Tstring);
 delete String;
 String = strdup(Tstring);
 if(2 <= Ndecimal && Ndecimal <= 5)
   Decimal = pow(10,Ndecimal);
 else
   Decimal = pow(10,2);
 Calcula();
}

void Dispersion::Proceso() {
 double Ivalor = 0.0;

 Ivalor = log( Suma_ASCII() ) * log10( pow(M_PI,lstring) );
 Valor = ObtenNm( ObtenNm(Ivalor) );
 modf(Valor,&Ivalor);
 Valor = Ivalor;
}

double Dispersion::ObtenNm(double Ptr) {
 double sptr = 0.0;
 modf(Ptr,&sptr); Ptr -= sptr;
 return ( Ptr *=Decimal );
}

double Dispersion::Suma_ASCII() {
 double suma = 0.0;
 int flag = 0;

 for(int i= 0; i < lstring; i++) {
  if(!flag)
    suma += sqrt( pow(M_E, NvCodigo(String[i]))), flag = 1;
  else
    suma += sqrt( pow(M_PI, NvCodigo(String[i]))), flag = 0;
 }

 return suma;
}

double Dispersion::NvCodigo(int Ctr) {
 int Cvalor = 0;
```

```
 if( int(Ctr) == 32) Cvalor = 99;
 else if( 33 <= int(Ctr) && int(Ctr) <= 57)
        Cvalor = int(Ctr) - 32;
 else if( 65 <= int(Ctr) && int(Ctr) <= 90)
        Cvalor = int(Ctr) - 39;
 else if( 97 <= int(Ctr) && int(Ctr) <= 122)
        Cvalor = int(Ctr) - 45;
 else {
        cout << "La llave posee caracteres que no se encuentran"
           << "disponibles dentro del Nuevo Codigo."
           << endl;
        exit(1);
 }

 return double(Cvalor/10.0);
}

void Dispersion::Calcula() {
 Proceso();
 AsignaNum( NumeroObt());
}

long const Dispersion::NumeroObt() const { return long(Valor) ; }

char const *Dispersion::NumeroFn() const { return NumeroBin(); }
```

Archivo cabecera BINTREE.H

```
# include <iostream.h>

class Node {
  public :
        // destructor
        virtual ~Node();
        // duplicador
        virtual Node* duplicate(void) const = 0;
        // comparacion de 2 objetos
        virtual int compare(Node const *other) const = 0;
        //funcion que opera sobre los datos dentro del nodo
        virtual void process(void) = 0;
        //Es llamada cuando un objeto es sumado en el arbol
        virtual void already_stored(void);
};

Node::~Node() { }

void Node::already_stored() {}

class Tree {
  public :
        // destructor, constructores
        ~Tree(void);
        Tree(void);
        Tree(Tree const &other);
        // asignacion
```

```
        Tree const &operator= (Tree const &other);
        // adicion de un nodo
        void add(Node *what);
        // procesando el orden en el arbol
        void preorder_walk(void);
        void inorder_walk(void);
        void postorder_walk(void);
  private :
        // primitivas
        void copy(Tree const &other);
        void destroy(void);
        //datos
        Tree *left,*right;
        Node *info;
};
```

Archivo fuente BINTREE.CPP

```
#Include "bintree.h"

// destructor : destruye el árbol
Tree::~Tree() {  destroy(); }

// constructor por default: inicializando en cero
Tree::Tree() {
 left = right = 0;
 info = 0;
}

// constructor copia : inicializa los contenidos de otros objetos
Tree::Tree(Tree const &other) {
 copy(other);
}

// asignacion sobrecargada
Tree const &Tree::operator= (Tree const &other) {
 if(this != &other)  {
  destroy();
  copy(other);
 }
 return (*this);
}

// sumando un nuevo nodo al árbol
void Tree::add(Node *what) {
 if(!info)
  info = what->duplicate();
 else  {
  register int
  cmp = (int) info->duplicate();
  if(cmp<0) {
   if(!left) {
        left = new Tree;
        left->info = what->duplicate();
   }
   else
```

```
        left->add(what);
    }
    else if(cmp>0) {
     if(!right) {
        right = new Tree;
        right->info = what->duplicate();
     }
     else
        right->add(what);
    }
    else
     info->already_stored();
  }
}

// procesando el orden el en árbol
void Tree::preorder_walk() {
 if(info)
   info->process();
 if(left)
   left->preorder_walk();
 if(right)
   right->preorder_walk();
}

void Tree::inorder_walk() {
 if(left)
   left->inorder_walk();
 if(info)
   info->process();
 if(right)
   right->inorder_walk();
}

void Tree::postorder_walk() {
 if(left)
   left->postorder_walk();
 if(right)
   right->postorder_walk();
 if(info)
   info->process();
}

// operaciones primitivas
void Tree::destroy() {
 delete info;
 if(left)
   delete left;
 if(right)
   delete right;
}

void Tree::copy(Tree const &other) {
 info=other.info ? other.info->duplicate() : 0;
 left=other.left ? new Tree(*other.left) : 0;
 right=other.right ? new Tree(*other.right) : 0;
}
```

Archivo cabecera APXPRINT.H

```
#if !defined(__apxprint_h)                    // Centinela de inclusión.
#define __apxprint_h

#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\printer.h>

class APXPrintOut : public TPrintout {
public:
  APXPrintOut (TPrinter *printer, const char far *title, TWindow* window,
               BOOL scale = TRUE) : TPrintout(title) {
      Printer = printer; Window = window; Scale = scale;
      MapMode = MM_ANISOTROPIC; }

  void GetDialogInfo (int& minPage, int& maxPage, int& selFromPage, int& selToPage);
  void BeginPrinting ();
  void BeginPage (TRect &clientR);
  void PrintPage (int page, TRect& rect, unsigned flags);
  void EndPage ();
  void SetBanding (BOOL b)      { Banding = b; }
  BOOL HasPage (int pageNumber);

protected:
  TWindow    *Window;
  BOOL       Scale;
  TPrinter   *Printer;
  int        MapMode;

  int        PrevMode;
  TSize      OldVExt, OldWExt;
  TRect      OrgR;
};

#Endif                                        // __apxprint_h - Fin del Centinela.
```

Archivo fuente APXPRINT.CPP

```
#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\listbox.h>
#include <owl\edit.h>
#include <owl\preview.h>

#include "apxprint.h"

// No permite establecer el rango de impresión en el dialogo de impresión, solamente la página
// dispuesta para ser impresa
void APXPrintOut::GetDialogInfo (int& minPage, int& maxPage, int& selFromPage, int& selToPage) {
  minPage = maxPage = 0;
  selFromPage = selToPage = 0;
```

```
}

void APXPrintOut::BeginPrinting () {
  TRect clientR;

  BeginPage(clientR);

  TFrameWindow *fWindow = TYPESAFE_DOWNCAST(Window, TFrameWindow);

  HFONT  hFont = (HFONT)fWindow->GetClientWindow()->GetWindowFont();
  TFont  font("Times New Roman", -12);
  if (hFont == 0)
   DC->SelectObject(font);
  else
   DC->SelectObject(TFont(hFont));

  TEXTMETRIC  tm;
  int fHeight = (DC->GetTextMetrics(tm) == TRUE) ? tm.tmHeight + tm.tmExternalLeading : 10;

  DC->RestoreFont();

  // ¿ Cuantas líneas de este font podemos incluir dentro de una página ?
  int linesPerPage = MulDiv(clientR.Height(), 1, fHeight);

  TPrintDialog::TData &printerData = Printer->GetSetup();

  int maxPg = 1;

  // Obtiene la ventana de la clase cliente ( estos son los contenidos que se van a imprimir)
  TEdit *clientEditWindow = 0;
  TListBox *clientListWindow = 0,

  clientEditWindow = TYPESAFE_DOWNCAST(fWindow-GetClientWindow(), TEdit);
  if (clientEditWindow)
    maxPg = ((clientEditWindow->GetNumLines() / linesPerPage) + 1.0);
  else {
    clientListWindow = TYPESAFE_DOWNCAST(fWindow->GetClientWindow(), TListBox);
    if (clientListWindow)
      maxPg = ((clientListWindow->GetCount() / linesPerPage) + 1.0);
  }

  // Computa el número de páginas a imprimir
  printerData.MinPage = 1;
  printerData.MaxPage = maxPg;

  EndPage();
  TPrintout::BeginPrinting();
}

void APXPrintOut::BeginPage (TRect &clientR) {
  TScreenDC screenDC;
  TSize screenRes(screenDC.GetDeviceCaps(LOGPIXELSX),
          screenDC.GetDeviceCaps(LOGPIXELSY));
  TSize printRes(DC->GetDeviceCaps(LOGPIXELSX),
          DC->GetDeviceCaps(LOGPIXELSY));

  // Temporalmente cambia el tamaño de la ventana (de tal manera que WM_PAINT y
```

```
    // sus consultas en el tamaño total de la ventana (GetClientRect) es el tamaño de la
    // ventana para la WM_PAINT de la ventana y la página de impresión cuando Paint es
    // llamado de la Página de impresión
    // Notese que no usamos AdjustWindowRect debido a que es más limitado y no acoplable.
    // En lugar de eso se calculan las diferencias en pixeles entre la ventana del cliente y la
    // ventana principal. Esta diferencia es entonces adicionada al clientRect para calcular el
    // tamaño de la nueva ventana principal para SetWindowPos.
    clientR = Window->GetClientRect();
    Window->MapWindowPoints(HWND_DESKTOP, (TPoint*)&clientR, 2);

    // Calcula extra X y Y pixeles para traer las dimensiones deuna ventana
    // cliente iguales a la ventana principal.
    OrgR = Window->GetWindowRect();
    int adjX = OrgR.Width() - clientR.Width();
    int adjY = OrgR.Height() - clientR.Height();

    // Escala condicional de DC para la ventana, por lo que la salida de impresión reensamblará.
    if (Scale) {
      clientR = Window->GetClientRect();
      PrevMode = DC->SetMapMode(MapMode);
      DC->SetViewportExt(PageSize, &OldVExt);

      // Escala la ventana para el tamaño de la página (Asume que left & top son 0).
      clientR.right = MulDiv(PageSize.cx, screenRes.cx, printRes.cx);
      clientR.bottom = MulDiv(PageSize.cy, screenRes.cy, printRes.cy);

      DC->SetWindowExt(clientR.Size(), &OldWExt);
    }

    // Calcula el tamaño de la ventana basada en las dimensiones (DC).
    // Cambia el tamaño de la ventana, avisa el cambio, orden y redibuja.
    // no son visibles los cambios a la ventana.
    Window->SetRedraw(FALSE);
    Window->SetWindowPos(0, 0, 0, clientR.Width() + adjX, clientR.Height() + adjY,
                         SWP_NOMOVE | SWP_NOREDRAW |
                         SWP_NOZORDER| SWP_NOACTIVATE);
}

void APXPrintOut::PrintPage (int page, TRect& bandRect, unsigned) {
  TRect clientR;

  BeginPage(clientR);

  if (Scale)
    DC->DPtoLP(bandRect, 2);
  // Cambia el rango de la impresora a la página actual.
  TPrintDialog::TData& printerData = Printer->GetSetup();
  int fromPg = printerData.FromPage;
  int toPg = printerData.ToPage;

  printerData.FromPage = page;
  printerData.ToPage = page;

  // Llama a la ventana para rellenar a si mismo al dispositivo de impresión.
  Window->Paint(*DC, FALSE, bandRect);

  printerData.FromPage = fromPg;
```

```
  printerData.ToPage = toPg;

  if (Scale)
    DC->LPtoDP(bandRect, 2);

  EndPage();
}

void APXPrintOut::EndPage () {
  // Cambia el tamaño al tamaño original de la ventana, ninguno es el visor.
  Window->SetWindowPos(0, 0, 0, OrgR.Width(), OrgR.Height(),
                       SWP_NOMOVE | SWP_NOREDRAW |
                       SWP_NOZORDER| SWP_NOACTIVATE);
  Window->SetRedraw(TRUE);

  // Restaura los cambios hechso en el DC
  if (Scale) {
    DC->SetWindowExt(OldWExt);
    DC->SetViewportExt(OldVExt);
    DC->SetMapMode(PrevMode);
  }
}

BOOL APXPrintOut::HasPage (int pageNumber) {
  TPrintDialog::TData &printerData = Printer->GetSetup();

  return (pageNumber >= printerData.MinPage) &&
         (pageNumber <= printerData.MaxPage);
}
```

Archivo *Cabecera* APXPREV.H

```
#if !defined(__apxprev_h)              // Centinela de inclusión.
#define __apxprev_h

#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\controlb.h>
#include <owl\printdia.h>
#include <owl\preview.h>

#include "apxprint.h"
#include "implsapp.rh"
//{{TDecoratedFrame = PreviewWindow}}
class PreviewWindow : public TDecoratedFrame {
public:
  PreviewWindow (TWindow *parentWindow, TPrinter *printer, TWindow* currWindow,
                 const char far* title, TLayoutWindow* client);
  ~PreviewWindow ();

  int          PageNumber;

  TWindow      *CurrWindow;
  TControlBar  *PreviewSpeedBar;
  TPreviewPage  *Page1;
```

```
    TPreviewPage   *Page2;
    TPrinter       *Printer;

    TPrintDC       *PrnDC;
    TSize          *PrintExtent;
    APXPrintOut    *Printout;

  private:
    TLayoutWindow  *Client;
    void SpeedBarState ();

  //{{PreviewWindowVIRTUAL_BEGIN}}
  protected:
    virtual void SetupWindow ();
  //{{PreviewWindowVIRTUAL_END}}

  //{{PreviewWindowRSP_TBL_BEGIN}}
  protected:
    void PPR_Previous ();
    void PPR_Next ();
    void PPR_OneUp ();
    void PPR_TwoUp ();
    void EvNCLButtonDown (UINT wHitTestCode, TPoint & point);
    void EvClose ();
  //{{PreviewWindowRSP_TBL_END}}
  DECLARE_RESPONSE_TABLE(PreviewWindow);
  };  //{{PreviewWindow}}

  #Endif                                    // __apxprev_h - Fin del Centinela.
```

Archivo fuente APXPREV.CPP

```
#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\controlb.h>
#include <owl\buttonga.h>
#include <owl\textgadg.h>
#include "apxprev.h"

#include "implsapp.rh"

//{{PreviewWindow - Implementación}}

DEFINE_RESPONSE_TABLE1(PreviewWindow, TDecoratedFrame)
  EV_COMMAND(APX_PPR_PREVIOUS, PPR_Previous),
  EV_COMMAND(APX_PPR_NEXT, PPR_Next),
  EV_COMMAND(APX_PPR_ONEUP, PPR_OneUp),
  EV_COMMAND(APX_PPR_TWOUP, PPR_TwoUp),
//{{PreviewWindowRSP_TBL_BEGIN}}
  EV_WM_NCLBUTTONDOWN,
  EV_WM_CLOSE,
//{{PreviewWindowRSP_TBL_END}}
END_RESPONSE_TABLE;

PreviewWindow::PreviewWindow ( TWindow *parentWindow, TPrinter *printer,
```

```
                                    TWindow* currWindow, const char far* title,
                                    TLayoutWindow* client) :
                                    TDecoratedFrame(parentWindow, title, client) {
  CurrWindow = currWindow;
  Printer = printer;
  Client = client;
  Page1 = 0;
  Page2 = 0;

  TPrintDialog::TData& data = Printer->GetSetup();
  PrnDC = new TPrintDC(data.GetDriverName(),
              data.GetDeviceName(),
              data.GetOutputName(),
              data.GetDevMode());

  PrintExtent = new TSize(PrnDC->GetDeviceCaps(HORZRES), PrnDC-GetDeviceCaps(VERTRES));
  Printout = new APXPrintOut(Printer, "Presentación previa", currWindow, TRUE);

  SetBkgndColor(GetSysColor(COLOR_APPWORKSPACE));

  // Crea por default la barra de herramientas y asocia los botones de la
  // barra de herrmientas con los comandos.
  PreviewSpeedBar = new TControlBar(this);
  PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_PREVIOUS, APX_PPR_PREVIOUS,
                          TButtonGadget::Command, TRUE));
  PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_NEXT, APX_PPR_NEXT,
                          TButtonGadget::Command, TRUE));
  PreviewSpeedBar->Insert(*new TSeparatorGadget(6));
  PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_ONEUP, APX_PPR_ONEUP,
                          TButtonGadget::Exclusive, TRUE, TButtonGadget::Down));
  PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_TWOUP, APX_PPR_TWOUP,
                          TButtonGadget::Exclusive, TRUE));
  PreviewSpeedBar->Insert(*new TSeparatorGadget(12));
  PreviewSpeedBar->Insert(*new TTextGadget(APX_PPR_CURRPAGE, TGadget::Recessed,
                          TTextGadget::Left, 10, "Pag. 1"));
  Insert(*PreviewSpeedBar, TDecoratedFrame::Top);

  // Nosotros deseamos una ventana que pueda ser cambiada de tamaño, es decir,
  // que pueda maximizarse y minimizarse.
  Attr.Style &= ~(WS_THICKFRAME | WS_BORDER | WS_MAXIMIZEBOX | WS_MINIMIZEBOX);
  Attr.Style |= (WS_VISIBLE | WS_POPUP | WS_CAPTION | WS_SYSMENU);

  // No muestra el borde de la ventana de la vista preeliminar.
  Attr.X = -1;
  Attr.Y = -1;
  Attr.W = Parent->GetClientRect().Width() + 2;
  Attr.H = Parent->GetClientRect().Height() + 2;
  parentWindow->MapWindowPoints(HWindow, (TPoint *)&(Attr.X), 1);
}

PreviewWindow::~PreviewWindow () {
  delete Page1;
  Page1 = 0;
  delete Page2;
  Page2 = 0;

  delete PrnDC;
```

```
    PrnDC = 0;
    delete PrintExtent;
    PrintExtent = 0;
    delete Printout;
    Printout = 0;
}

void PreviewWindow::SetupWindow () {
    TDecoratedFrame::SetupWindow();
    TPrintDialog::TData& data = Printer->GetSetup();
    Page1 = new TPreviewPage(Client, *Printout, *PrnDC, *PrintExtent, 1);
    Page1->SetPageNumber(1);
    data.FromPage = 1;
    data.ToPage = 1;
    data.MinPage = 1;
    data.MaxPage = 1;

    Page2 = 0;

    TLayoutMetrics metrics1;

    metrics1.X.Set(lmLeft, lmRightOf, lmParent, lmLeft, 15);
    metrics1.Y.Set(lmTop, lmBelow, lmParent, lmTop, 15);

    // Determina la mayor absisa de la página de la vista preliminar, tiene la siguiente un tamaño parecido.
    // Hace una absisa menor en porcentaje() de la página de mayor absisa.
    if (PrintExtent->cx  > PrintExtent->cy)  {
        metrics1.Width.Set(lmRight, lmLeftOf, lmParent, lmRight, 15);
        metrics1.Height.PercentOf(Page1, int((long(PrintExtent->cy) * 100) / PrintExtent->cx), lmWidth);

        // Dos páginas hacia arriba, solamente disponible para el formato vertical.
        TGadget *theGadget = PreviewSpeedBar->GadgetWithId(APX_PPR_TWOUP);
        theGadget->SetEnabled(FALSE);
    } else {
        metrics1.Height.Set(lmBottom, lmAbove, lmParent, lmBottom, 15);
        metrics1.Width.PercentOf(Page1, int((long(PrintExtent->cx) * 95) / PrintExtent->cy), lmHeight);
    }

    TGadget *theGadget = PreviewSpeedBar->GadgetWithId(APX_PPR_PREVIOUS);
    theGadget->SetEnabled(FALSE);

    Page1->Create();

    Client->SetChildLayoutMetrics(*Page1, metrics1);
    Client->Layout();
}

void PreviewWindow::SpeedBarState () {
    TPrintDialog::TData &printerData = Printer->GetSetup();

    // Solamente tiene vista previa, si no se encuentra en la primera página.
    TGadget *theGadget = PreviewSpeedBar->GadgetWithId(APX_PPR_PREVIOUS);
    theGadget->SetEnabled(printerData.FromPage != 1);

    // Solamente tiene avance, si no se encuentra en la última hoja.
    theGadget = PreviewSpeedBar->GadgetWithId(APX_PPR_NEXT);
```

```
  theGadget->SetEnabled(printerData.ToPage != printerData.MaxPage);

  // Actualiza el contador de las páginas.
  TTextGadget *theTGadget = TYPESAFE_DOWNCAST(
                            PreviewSpeedBar->GadgetWithId(APX_PPR_CURRPAGE),
                            TTextGadget);
  if (theTGadget) {
    char  buffer[32];
    if (Page2 && (printerData.FromPage != printerData.ToPage))
      wsprintf(buffer, "Pag. %d - %d", printerData.FromPage, printerData.ToPage);
    else
      wsprintf(buffer, "Pag. %d", printerData.FromPage);
    theTGadget->SetText(buffer);
  }
}

void PreviewWindow::EvClose () {
  // No llama a la claseEvClose, no se desea que la vista preeliminar se destruya.
  GetApplication()->EndModal(IDCANCEL);
}

void PreviewWindow::PPR_Previous () {
  TPrintDialog::TData &printerData = Printer->GetSetup();

  if (printerData.FromPage  printerData.MinPage) {
    printerData.FromPage—;
    printerData.ToPage—;

    Page1->SetPageNumber(printerData.FromPage);
    if (Page2)
      Page2->SetPageNumber(printerData.ToPage);
  }

  SpeedBarState();
}


void PreviewWindow::PPR_Next () {
  TPrintDialog::TData &printerData = Printer->GetSetup();

  if (printerData.ToPage printerData.MaxPage) {
    printerData.FromPage++;
    printerData.ToPage++;

    Page1->SetPageNumber(printerData.FromPage);
    if (Page2)
      Page2->SetPageNumber(printerData.ToPage);
  }

  SpeedBarState();
}

void PreviewWindow::PPR_OneUp () {
  if (Page2) {
    Client->RemoveChildLayoutMetrics(*Page2);

    delete Page2;
```

```
    Page2 = 0;

    Client->Layout();

    TPrintDialog::TData &printerData = Printer->GetSetup();
    printerData.ToPage = printerData.FromPage;

    SpeedBarState();
  }
}

void PreviewWindow::PPR_TwoUp () {
  if (Page2 == 0) {
    Page2 = new TPreviewPage(Client, *Printout, *PrnDC, *PrintExtent, PageNumber + 1);
    Page2->Create();

    TLayoutMetrics metrics2;

    metrics2.X.Set(lmLeft, lmRightOf, Page1, lmRight, 30);
    metrics2.Y.SameAs(Page1, lmTop);

    // Asume vertical.
    metrics2.Width.SameAs(Page1, lmWidth);
    metrics2.Height.SameAs(Page1, lmBottom);

    Client->SetChildLayoutMetrics(*Page2, metrics2);
    Client->Layout();

    TPrintDialog::TData &printerData = Printer->GetSetup();

    // La pagina 2 es la página siguiente. Si la siguiente página se encuentra fuera de nuestro
    // rango, emtonces el conjunto de la primera página uno y la segunda página es la
    // la página actual.  Si el documento es solamente de una página de longitud
    // la segunda pagina se encuentra vacía.
    if (printerData.FromPage == printerData.MaxPage) {
      if (printerData.FromPage > 1) {
        printerData.FromPage--;
        printerData.ToPage = printerData.FromPage + 1;
        Page1->SetPageNumber(printerData.FromPage);
        Page2->SetPageNumber(printerData.ToPage);
      } else
        Page2->SetPageNumber(0);
    } else {
      printerData.ToPage = printerData.FromPage + 1;
      Page2->SetPageNumber(printerData.ToPage);
    }

    SpeedBarState();
  }
}

// No permite la ventana de vista preeliminar para ser tomada alrededor.
void PreviewWindow::EvNCLButtonDown(UINT wHitTestCode, TPoint & point) {
  // Si el LButtonDown no se encuentra la señal , entonces el proceso es normal.
  if (wHitTestCode != HTCAPTION)
    TDecoratedFrame::EvNCLButtonDown(wHitTestCode, point);
}
```

Archivo *cabecera* IMPLMDIC.H

```
#if !defined(__implmdic_h)              // Centinela de inclusión.
#define __implmdic_h

#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\opensave.h>

#include "implsapp.rh"                  // Definición de todos los recursos.


//{{TMDIClient = ImpulsoMDIClient}}
class ImpulsoMDIClient : public TMDIClient {
public:
  int           ChildCount;             // Número de ventanas hijas creadas.

  ImpulsoMDIClient ();
  virtual ~ImpulsoMDIClient ();

  void OpenFile (const char *fileName = 0);

private:
  void LoadTextFile ();

//{{ImpulsoMDIClientVIRTUAL_BEGIN}}
protected:
  virtual void SetupWindow ();
//{{ImpulsoMDIClientVIRTUAL_END}}

//{{ImpulsoMDIClientRSP_TBL_BEGIN}}
protected:
  void CmFilePrint ();
  void CmFilePrintSetup ();
  void CmFilePrintPreview ();
  void CmPrintEnable (TCommandEnabler &tce);
  void EvDropFiles (TDropInfo);
//{{ImpulsoMDIClientRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(ImpulsoMDIClient);
};  //{{ImpulsoMDIClient}}

#Endif                                  // __implmdic_h - Fín del Centinela.
```

Archivo *fuente* IMPLMDIC.CPP

```
#include <owl\owlpch.h>
#pragma hdrstop

#include <dir.h>

#include "implsapp.h"
#include "implmdic.h"
#include "implmdi1.h"
#include "apxprint.h"
```

```
#include "apxprev.h"

//{{ImpulsoMDIClient - Implementación}}

// Construye una tabla responsable para todos los mensajes/comandos manejados
// por ImpulsoMDIClient derivado de TMDIClient.

DEFINE_RESPONSE_TABLE1(ImpulsoMDIClient, TMDIClient)
//{{ImpulsoMDIClientRSP_TBL_BEGIN}}
  EV_COMMAND(CM_FILEPRINT, CmFilePrint),
  EV_COMMAND(CM_FILEPRINTERSETUP, CmFilePrintSetup),
  EV_COMMAND(CM_FILEPRINTPREVIEW, CmFilePrintPreview),
  EV_COMMAND_ENABLE(CM_FILEPRINT, CmPrintEnable),
  EV_COMMAND_ENABLE(CM_FILEPRINTERSETUP, CmPrintEnable),
  EV_COMMAND_ENABLE(CM_FILEPRINTPREVIEW, CmPrintEnable),
  EV_WM_DROPFILES,
//{{ImpulsoMDIClientRSP_TBL_END}}
END_RESPONSE_TABLE;

// Manejador del Constructor/Destructor.
ImpulsoMDIClient::ImpulsoMDIClient () : TMDIClient () {
  // Cambia el color de fondo de la ventana principal.
  SetBkgndColor(RGB(0x00, 0x80, 0x80));

  ChildCount = 0;
}

ImpulsoMDIClient::~ImpulsoMDIClient () {   Destroy(); }

// Sitio de inicialización de MDIClient.
void ImpulsoMDIClient::SetupWindow () {
  // Proceso por default para SetUpWindow.
  TMDIClient::SetupWindow ();

  // Acepta archivoss via arrastrar/soltar en la ventana principal..
  DragAcceptFiles(TRUE);
}



// Menú - Archivo. Comando Imprimir.
void ImpulsoMDIClient::CmFilePrint () {
  // Crea el objeto de impresión y el conjunto de características.
  ImpulsoApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), ImpulsoApp);
  if (theApp) {
    if (!theApp->Printer)
      theApp->Printer = new TPrinter;

    // Crea la ventana de impresión y el cojunto de características.
    APXPrintOut printout(theApp->Printer, Title, GetActiveMDIChild(), TRUE);

    theApp->Printing = TRUE;

    // Se establece la comunicación entre la impresora y la impresión del documento.
    theApp->Printer->Print(GetActiveMDIChild()->GetClientWindow(), printout, TRUE);

    theApp->Printing = FALSE;
```

```
  }
}

// Menú - Archivo. Comando Especificar impresora.
void ImpulsoMDIClient::CmFilePrintSetup () {
  ImpulsoApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), ImpulsoApp);
  if (theApp) {
    if (!theApp->Printer)
      theApp->Printer = new TPrinter;

    // Carag el dialogo de configuración de la impresora.
    theApp->Printer->Setup(this);
  }
}

// Menú- Archivo. Comando Presentación previa ... con opción de imprimir.
void ImpulsoMDIClient::CmFilePrintPreview () {
  ImpulsoApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), ImpulsoApp);
  if (theApp) {
    if (!theApp-Printer)
      theApp->Printer = new TPrinter;

    theApp->Printing = TRUE;

    PreviewWindow *prevW = new PreviewWindow(Parent, theApp-Printer, GetActiveMDIChild(),
                                             "Presentación previa", new TLayoutWindow(0));
    prevW->Create();

    GetApplication()->BeginModal(GetApplication()->MainWindow);

    // Debemos destruir la ventana preeliminar explicitamente.En otro caso, la ventana
    // no será destruida, hasta que la ventana principalaparente sea destruída.
    prevW->Destroy();
    delete prevW;

    theApp->Printing = FALSE;
  }
}

// Menú habilitador usado para imprimir, configurar impresora y presentación previa.
void ImpulsoMDIClient::CmPrintEnable (TCommandEnabler &tce) {
  if (GetActiveMDIChild()) {
    ImpulsoApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), ImpulsoApp);
    if (theApp) {
      // Si tenemos una impresora ya creada, esta prueba es para checar si esta Ok.
      // En otro caso, crea el objeto de impresión y asegura que la impresora realmente exista
      // y entonces elimina el objeto de la impresión.
      if (!theApp->Printer) {
        theApp->Printer = new TPrinter;
        tce.Enable(theApp->Printer-GetSetup().Error == 0);
      } else
        tce.Enable(theApp->Printer-GetSetup().Error == 0);
    }
  } else
    tce.Enable(FALSE);
}
```

```
void ImpulsoMDIClient::EvDropFiles (TDropInfo) { Parent->ForwardMessage(); }
```

Archivo cabecera IMPLMDI1.H

```
#if !defined(__implmdi1_h)                    // Centinela de inclusión.
#define __implmdi1_h

#include <owl\editfile.h>
#include <owl\listbox.h>

#include "implsapp.rh"                        // Definición de todos los recursos.
//{{TMDIChild = ImpulsoMDIChild}}
class ImpulsoMDIChild : public TMDIChild {
public:
  ImpulsoMDIChild (TMDIClient &parent, const char far *title, TWindow *clientWnd,
                   BOOL shrinkToClient = FALSE, TModule* module = 0);
  virtual ~ImpulsoMDIChild ();

//{{ImpulsoMDIChildVIRTUAL_BEGIN}}
public:
  virtual void Paint (TDC& dc, BOOL erase, TRect& rect);
//{{ImpulsoMDIChildVIRTUAL_END}}
//{{ImpulsoMDIChildRSP_TBL_BEGIN}}
protected:
  void EvGetMinMaxInfo (MINMAXINFO far& minmaxinfo);
//{{ImpulsoMDIChildRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(ImpulsoMDIChild);
};  //{{ImpulsoMDIChild}}

#Endif                                        // __implmdi1_h - Fín del Centinela.
```

Archivo fuente IMPLMDI1.CPP

```
#include <owl\owlpch.h>
#pragma hdrstop

#include "implsapp.h"
#include "implmdi1.h"

#include <stdio.h>

//{{ImpulsoMDIChild Implementation}}

// Construye una tabla de respuestas para todos los mensajes/comandos manejados
// por ImpulsoMDIChild derivados de TMDIChild.

DEFINE_RESPONSE_TABLE1(ImpulsoMDIChild, TMDIChild)
//{{ImpulsoMDIChildRSP_TBL_BEGIN}}
  EV_WM_GETMINMAXINFO,
//{{ImpulsoMDIChildRSP_TBL_END}}
END_RESPONSE_TABLE;

// Manejador del Constructor/Destructor.
ImpulsoMDIChild::ImpulsoMDIChild (TMDIClient &parent, const char far *title,
                                  TWindow *clientWnd, BOOL shrinkToClient,
                                  TModule *module)
```

```
                                : TMDIChild (parent, title, clientWnd,
                                             shrinkToClient, module) { }

ImpulsoMDIChild::~ImpulsoMDIChild () { Destroy(); }

// Rutina de dibujo para la ventana, impresora y vista preeliminar para un Cliente TEdit.
void ImpulsoMDIChild::Paint (TDC& dc, BOOL, TRect& rect) {
  ImpulsoApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), ImpulsoApp);
  if (theApp) {
    // Solamente pinta si se encuentra la impresora y se tiene algo para pintar, en otro caso no hace nada.
    if (theApp->Printing && theApp->Printer && !rect.IsEmpty()) {
      // Utiliza el tamaño de la página para obtener el tamaño de la ventana que
      // contendrá la información. Para una ventana es el área del cliente, para una impresora,
      // las dimensiones de la impresora de DC y para la vista preeliminar se encuentran en la
      // ventana de la vista.
      TSize  pageSize(rect.right - rect.left, rect.bottom - rect.top);

      HFONT  hFont = (HFONT)GetClientWindow()->GetWindowFont();
      TFont  font("Times New Roman", -10);
      if (hFont == 0)
       dc.SelectObject(font);
      else
       dc.SelectObject(TFont(hFont));

      TEXTMETRIC  tm;
      int fHeight = (dc.GetTextMetrics(tm) == TRUE) ? tm.tmHeight + tm.tmExternalLeading : 10;

      // Cuantas lineas de este font pueden entrar en esta página ?
      int linesPerPage = MulDiv(pageSize.cy, 1, fHeight);
      if (linesPerPage) {
        TPrintDialog::TData &printerData = theApp->Printer-GetSetup();

        int maxPg = 1;

        // Obtiene la ventana de la clase cliente (Este es el contenido que vamos a imprimir).
        TEdit *clientEditWindow = 0;
        TListBox *clientListWindow = 0;
        clientEditWindow = TYPESAFE_DOWNCAST(GetClientWindow(), TEdit);
        if (clientEditWindow)
          maxPg = ((clientEditWindow->GetNumLines() / linesPerPage) + 1.0);
        else {
          clientListWindow = TYPESAFE_DOWNCAST(GetClientWindow(), TListBox);
          if (clientListWindow)
            maxPg = ((clientListWindow->GetCount() / linesPerPage) + 1.0); }
        // Calcula el número de páginas a imprimir.
        printerData.MinPage = 1;
        printerData.MaxPage = maxPg;

        // Hace el siguiente paso.
        int    fromPage = printerData.FromPage == -1 ? 1 : printerData.FromPage;
        int    toPage = printerData.ToPage == -1 ? 1 : printerData.ToPage;
        char   buffer[255];
        int    currentPage = fromPage;

        while (currentPage <= toPage) {
          int startLine = (currentPage - 1) * linesPerPage;
          int lineIdx = 0;
```

```
            while (lineIdx linesPerPage) {
              // Si la cadena no es de una longitud valida, entonces no hay más que desplegar.
              if (clientEditWindow) {
                if (!clientEditWindow->GetLine(buffer, sizeof(buffer), startLine + lineIdx))
                  break;
              }
              if (clientListWindow) {
                if (clientListWindow->GetString(buffer, startLine + lineIdx) 0)
                  break;
              }
              dc.TabbedTextOut(TPoint(0, lineIdx * fHeight), buffer, lstrlen(buffer), 0, NULL, 0);
              lineIdx++;
            }
            currentPage++;
          }
        }
      }
    }
  }

void ImpulsoMDIChild::EvGetMinMaxInfo (MINMAXINFO far& minmaxinfo) {
  ImpulsoApp *theApp = TYPESAFE_DOWNCAST(GetApplication(), ImpulsoApp);
  if (theApp) {
    if (theApp->Printing) {
      minmaxinfo.ptMaxSize = TPoint(32000, 32000);
      minmaxinfo.ptMaxTrackSize = TPoint(32000, 32000);
      return;
    }
  }
  TMDIChild::EvGetMinMaxInfo(minmaxinfo);
}
```

Archivo cabecera IMPLSABD.H

```
#if !defined(__implsabd_h)              // Centinela de inclusión.
#define __implsabd_h

#include <owl\owlpch.h>
#pragma hdrstop

#include "implsapp.rh"                  // Definición de todos los recursos.

//{{TDialog = ImpulsoAboutDlg}}
class ImpulsoAboutDlg : public TDialog {
public:
  ImpulsoAboutDlg (TWindow *parent, TResId resId = IDD_ABOUT, TModule *module = 0);
  virtual ~ImpulsoAboutDlg ();

//{{ImpulsoAboutDlgVIRTUAL_BEGIN}}
public:
  void SetupWindow ();
//{{ImpulsoAboutDlgVIRTUAL_END}}
};   //{{ImpulsoAboutDlg}}

#Endif                                  // __implsabd_h - Fin del Centinela.
```

Archivo *cabecera* IMPLSABD.CPP

```
#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\static.h>

#include <ver.h>

#include "implsapp.h"
#include "implsabd.h"


// Leyendo la información respectiva al proceso VERSIONINFO de recursos.
class ProjectRCVersion {
public:
  ProjectRCVersion (TModule *module);
  virtual ~ProjectRCVersion ();

  BOOL GetProductName (LPSTR &prodName);
  BOOL GetProductVersion (LPSTR &prodVersion);
  BOOL GetCopyright (LPSTR &copyright);
  BOOL GetDebug (LPSTR &debug);

protected:
  LPBYTE    TransBlock;
  void FAR  *FVData;

private:
  // No permite a este objeto ser copiado.
  ProjectRCVersion (const ProjectRCVersion &);
  ProjectRCVersion & operator =(const ProjectRCVersion &);
};


ProjectRCVersion::ProjectRCVersion (TModule *module) {
  char  appFName[255];
  DWORD  fvHandle;
  UINT  vSize;

  FVData = 0;

  module->GetModuleFileName(appFName, sizeof(appFName));
  DWORD dwSize = GetFileVersionInfoSize(appFName, &fvHandle);
  if (dwSize) {
    FVData  = (void FAR *)new char[(UINT)dwSize];
    if (GetFileVersionInfo(appFName, fvHandle, dwSize, FVData))
      if (!VerQueryValue(FVData, "\\VarFileInfo\\Translation",
                         (void FAR* FAR*)&TransBlock, &vSize)) {
        delete FVData;
        FVData = 0;
      }
  }
}
```

```
ProjectRCVersion::~ProjectRCVersion () {
  if (FVData)
    delete FVData;
}

BOOL ProjectRCVersion::GetProductName (LPSTR &prodName) {
  UINT   vSize;
  char   subBlockName[255];

  wsprintf(subBlockName, "\\StringFileInfo\\%08lx\\%s", *(DWORD *)TransBlock,
          (LPSTR)"ProductName");
  return FVData ? VerQueryValue(FVData, subBlockName,
                                (void FAR* FAR*)&prodName, &vSize) : FALSE;
}

BOOL ProjectRCVersion::GetProductVersion (LPSTR &prodVersion) {
  UINT   vSize;
  char   subBlockName[255];

  wsprintf(subBlockName, "\\StringFileInfo\\%08lx\\%s", *(DWORD *)TransBlock,
          (LPSTR)"ProductVersion");
  return FVData ? VerQueryValue(FVData, subBlockName,
                                (void FAR* FAR*)&prodVersion, &vSize) : FALSE;
}

BOOL ProjectRCVersion::GetCopyright (LPSTR &copyright) {
  UINT   vSize;
  char   subBlockName[255];

  wsprintf(subBlockName, "\\StringFileInfo\\%08lx\\%s", *(DWORD *)TransBlock,
          (LPSTR)"LegalCopyright");
  return FVData ? VerQueryValue(FVData, subBlockName,
                                (void FAR* FAR*)&copyright, &vSize) : FALSE;
}


BOOL ProjectRCVersion::GetDebug (LPSTR &debug) {
  UINT   vSize;
  char   subBlockName[255];

  wsprintf(subBlockName, "\\StringFileInfo\\%08lx\\%s", *(DWORD *)TransBlock,
          (LPSTR)"SpecialBuild");
  return FVData ? VerQueryValue(FVData, subBlockName, (void FAR* FAR*)&debug,
                          &vSize) : FALSE;
}

//{{ImpulsoAboutDlg - Iimplementación}}

// Manejador del Constructor/Destructor.
ImpulsoAboutDlg::ImpulsoAboutDlg (TWindow *parent, TResId resId, TModule *module)
  : TDialog(parent, resId, module) { }

ImpulsoAboutDlg::~ImpulsoAboutDlg () { Destroy(); }

void ImpulsoAboutDlg::SetupWindow () {
  LPSTR prodName, prodVersion, copyright, debug;
```

```
  // Obtiene el texto estático cuyo valor se encuentra basado en VERSIONINFO.
  TStatic *versionCtrl = new TStatic(this, IDC_VERSION, 255);
  TStatic *copyrightCtrl = new TStatic(this, IDC_COPYRIGHT, 255);
  TStatic *debugCtrl = new TStatic(this, IDC_DEBUG, 255);

  TDialog::SetupWindow();

  // Procesamiento de VERSIONINFO.
  ProjectRCVersion applVersion(GetModule());

  // Obtiene el nombre del producto, la versión y la cadena del certificado de copia legal.
  applVersion.GetProductName(prodName);
  applVersion.GetProductVersion(prodVersion);
  applVersion.GetCopyright(copyright);

  // IDC_VERSION Es el nombre dle producto, el número de la versión y el valor inicial de
  // IDC_VERSION, es la versión en palabras en cualquiera que se el lenguaje del nombre del producto
  // VERSION y la version del mismo.
  char  buffer[255];
  char  versionName[128];
  versionCtrl->GetText(versionName, sizeof(versionName));
  wsprintf(buffer, "%s %s %s", prodName, versionName, prodVersion);
  versionCtrl->SetText(buffer);

  copyrightCtrl->SetText(copyright);

  // Solamente obtiene el proceso SpecialBuild del texto si el proceso VERSIONINFO devuelve la
  // información de los recusroso.
  if (applVersion.GetDebug(debug))
    debugCtrl->SetText(debug);
}
```

Archivo *cabecera* IMPLSAPP.RH

```
//#if !defined(__implsapp_rh)
//#define __implsapp_rh

// IDHELP BorButton para dialogos de BWCC.
#define IDHELP                  998         // Identificación del boton de ayuda.

// Definiciones de aplicación especifica.
#define IDI_MDIAPPLICATION      1001        // Icono de la  aplicación.
#define IDI_DOC                 1002        // Icono de la ventana hija MDI.

#define MDI_MENU                100         // Menu de recursos y aceleradores IDs

// Comandos CM_FILEnnnn (include\owl\editfile.rh excepto para CM_FILEPRINTPREVIEW)
#define CM_MDIFILENEW           24331
#define CM_MDIFILEOPEN          24332
#define CM_FILECLOSE            24339
#define CM_FILESAVE             24333
#define CM_FILESAVEAS           24334
#define CM_FILEREVERT           24335
```

```
#define CM_VIEWCREATE                24341
#define CM_FILEPRINT                 24337
#define CM_FILEPRINTERSETUP          24338
#define CM_FILEPRINTPREVIEW          24340

// Comandos para ventana (include\owl\windows.rh)
#define CM_EXIT                      24310

// ComandosCM_EDITnnnn  (include\owl\edit.rh)
#define CM_EDITUNDO                  24321
#define CM_EDITCUT                   24322
#define CM_EDITCOPY                  24323
#define CM_EDITPASTE                 24324
#define CM_EDITDELETE                24325
#define CM_EDITCLEAR                 24326

// Comandos del menu de Búsqueda (include\owl\editsear.rh)
#define CM_EDITFIND                  24351
#define CM_EDITREPLACE               24352
#define CM_EDITFINDNEXT              24353

// Comandos del sistema en operación (include\owl\mdi.rh)
#define CM_CASCADECHILDREN           24361
#define CM_TILECHILDREN              24362
#define CM_TILECHILDRENHORIZ         24363
#define CM_ARRANGEICONS              24364
#define CM_CLOSECHILDREN             24365
#define CM_CREATECHILD               24366

// Comandos del menú de Ayuda.
#define CM_HELPCONTENTS              24381
#define CM_HELPUSING                 24382
#define CM_HELPABOUT                 24389


// Contexto sensitivo del cursor de ayuda.
#define IDC_HELPCURSOR               24000

// Dialogo Acerca de ...
#define IDD_ABOUT                    22000
#define IDC_VERSION                  22001
#define IDC_COPYRIGHT                22002
#define IDC_DEBUG                    22003

// Cadenas definidas dentro del OWL

// Barra de Estado
#define IDS_MODES                    32530

// Editor de Archivo
#define IDS_UNABLEREAD               32551
#define IDS_UNABLEWRITE              32552
#define IDS_FILECHANGED              32553
#define IDS_FILEFILTER               32554

// Editor de Búsqueda
#define IDS_CANNOTFIND               32540
```

```
// Mensajes generales y de aplicaciones de excepcioness (include\owl\except.rh)
#define IDS_UNKNOWNEXCEPTION    32767
#define IDS_OWLEXCEPTION        32766
#define IDS_OKTORESUME          32765
#define IDS_UNHANDLEDXMSG       32764
#define IDS_UNKNOWNERROR        32763
#define IDS_NOAPP               32762
#define IDS_OUTOFMEMORY         32761
#define IDS_INVALIDMODULE       32760
#define IDS_INVALIDMAINWINDOW   32759

// Mensajes de compatibilidad del Owl 1
#define IDS_INVALIDWINDOW       32756
#define IDS_INVALIDCHILDWINDOW  32755
#define IDS_INVALIDCLIENTWINDOW 32754

// Mensajes de error del TXWindow
#define IDS_CLASSREGISTERFAIL   32749
#define IDS_CHILDREGISTERFAIL   32748
#define IDS_WINDOWCREATEFAIL    32747
#define IDS_WINDOWEXECUTEFAIL   32746
#define IDS_CHILDCREATEFAIL     32745

#define IDS_MENUFAILURE         32744
#define IDS_VALIDATORSYNTAX     32743
#define IDS_PRINTERERROR        32742

#define IDS_LAYOUTINCOMPLETE    32741
#define IDS_LAYOUTBADRELWIN     32740

// Mensajes de error del TXGdi
#define IDS_GDIFAILURE          32739
#define IDS_GDIALLOCFAIL        32738
#define IDS_GDICREATEFAIL       32737
#define IDS_GDIRESLOADFAIL      32736
#define IDS_GDIFILEREADFAIL     32735
#define IDS_GDIDELETEFAIL       32734
#define IDS_GDIDESTROYFAIL      32733
#define IDS_INVALIDDIBHANDLE    32732

// DocView (include\owl\docview.rc)
#define IDS_DOCMANAGERFILE      32500
#define IDS_DOCLIST             32501
#define IDS_VIEWLIST            32502
#define IDS_UNTITLED            32503
#define IDS_UNABLEOPEN          32504
#define IDS_UNABLECLOSE         32505
#define IDS_READERROR           32506
#define IDS_WRITEERROR          32507
#define IDS_DOCCHANGED          32508
#define IDS_NOTCHANGED          32509
#define IDS_NODOCMANAGER        32510
#define IDS_NOMEMORYFORVIEW     32511
#define IDS_DUPLICATEDOC        32512

// Error de impresión de recusros de cadena en mensajes IDs (include\owl\printer.rh)
```

```
#define IDS_PRNON                    32590
#define IDS_PRNERRORTEMPLATE         32591
#define IDS_PRNOUTOFMEMORY           32592
#define IDS_PRNOUTOFDISK             32593
#define IDS_PRNCANCEL                32594
#define IDS_PRNMGRABORT              32595
#define IDS_PRNGENERROR              32596
#define IDS_PRNERRORCAPTION          32597

// Dialogo de la cancelación de impresión y control de IDs
#define IDD_ABORTDIALOG              32599
#define ID_DEVICE                    102
#define ID_PORT                      103

// Presentación previa
#define APX_PPR_PREVIOUS             24500
#define APX_PPR_NEXT                 24501
#define APX_PPR_ONEUP                24502
#define APX_PPR_TWOUP                24503
#define APX_PPR_CURRPAGE             24504

// Recurso del DIALOG TInputDialog  (include\owl\inputdia.rh)
#define IDD_INPUTDIALOG              32514
#define ID_PROMPT                    4091
#define ID_INPUT                     4090

// bitmaps TSlider  (horizontales y verticales) (include\owl\slider.rh)
#define IDB_HSLIDERTHUMB             32000
#define IDB_VSLIDERTHUMB             32001

// Mensajes de validación (include\owl\validate.rh)
#define IDS_VALPXPCONFORM            32520
#define IDS_VALINVALIDCHAR           32521
#define IDS_VALNOTINRANGE            32522
#define IDS_VALNOTINLIST             32523

//#Endif
```

Archivo fuente IMPLSAPP.RC

```
#if !defined(WORKSHOP_INVOKED)
#include <windows.h>
#endif
#include "implsapp.rh"

MDI_MENU MENU
BEGIN
 POPUP "&Archivo"
 BEGIN
  MENUITEM "&Nuevo ...", CM_MDIFILENEW
  MENUITEM "&Abrir ...", CM_MDIFILEOPEN
  MENUITEM "&Cerrar", CM_FILECLOSE
  MENUITEM SEPARATOR
  MENUITEM "&Guardar\tF2", CM_FILESAVE, GRAYED
  MENUITEM "G&uardar como...", CM_FILESAVEAS, GRAYED
  MENUITEM SEPARATOR
```

```
 MENUITEM "&Presentación previa ...", CM_FILEPRINTPREVIEW, GRAYED
 MENUITEM "&Imprimir ...\tCtrl+P", CM_FILEPRINT, GRAYED
 MENUITEM "&Especificar impresora...", CM_FILEPRINTERSETUP, GRAYED
 MENUITEM SEPARATOR
 MENUITEM "&Salir\tAlt+F4", CM_EXIT
END

POPUP "&Edición"
BEGIN
 MENUITEM "&Deshacer\tCtrl+Z", CM_EDITUNDO, GRAYED
 MENUITEM SEPARATOR
 MENUITEM "C&ortar\tCtrl+X", CM_EDITCUT, GRAYED
 MENUITEM "&Copiar\tCtrl+C", CM_EDITCOPY, GRAYED
 MENUITEM "&Pegar\tCtrl+V", CM_EDITPASTE, GRAYED
 MENUITEM SEPARATOR
 MENUITEM "&Limpiar todo\tCtrl+Supr", CM_EDITCLEAR, GRAYED
 MENUITEM "&Borrar\tSupr", CM_EDITDELETE, GRAYED
END

POPUP "&Buscar"
BEGIN
 MENUITEM "&Buscar ...", CM_EDITFIND, GRAYED
 MENUITEM "&Reemplazar ...", CM_EDITREPLACE, GRAYED
 MENUITEM "&Siguiente\tF3", CM_EDITFINDNEXT, GRAYED
END

POPUP "&Ventana"
BEGIN
 MENUITEM "&Cascada", CM_CASCADECHILDREN
 MENUITEM "&Mosaico", CM_TILECHILDREN
 MENUITEM "Ordenar iconos", CM_ARRANGEICONS
 MENUITEM "Cerrar &todo", CM_CLOSECHILDREN
END
POPUP "&?"
BEGIN
 MENUITEM "&Contenido", CM_HELPCONTENTS
 MENUITEM "&Uso de la ayuda", CM_HELPUSING
 MENUITEM SEPARATOR
 MENUITEM "&Acerca ...", CM_HELPABOUT
END

END

// Tabla de aceleradores para accesos directos del menú de comandos. (include\owl\editfile.rc)
MDI_MENU ACCELERATORS
BEGIN
 VK_X, CM_EDITCUT, VIRTKEY, CONTROL
 VK_C, CM_EDITCOPY, VIRTKEY, CONTROL
 VK_V, CM_EDITPASTE, VIRTKEY, CONTROL
 VK_DELETE, CM_EDITCLEAR, VIRTKEY, CONTROL
 VK_Z, CM_EDITUNDO, VIRTKEY, CONTROL
 VK_F3, CM_EDITFINDNEXT, VIRTKEY
 VK_P, CM_FILEPRINT, VIRTKEY, CONTROL
 VK_F2, CM_FILESAVE, VIRTKEY
END

// Contexto sensitivo del cursore de ayuda..
```

```
IDC_HELPCURSOR CURSOR "help.cur"

// Tabla de ayuda que se encuentra desplegada dentro de la barra de estado.
STRINGTABLE
BEGIN
 CM_MDIFILENEW -1, "Operaciones de Archivos/Documento "
 CM_MDIFILENEW, "Crea un nuevo documento"
 CM_MDIFILEOPEN, "Abre un documento existente"
 CM_VIEWCREATE, "Crea una nueva vista para el documento"
 CM_FILEREVERT, "Deshace los cambios en la última salvada del documento"
 CM_FILECLOSE, "Cierra el documento activo"
 CM_FILESAVE, "Guarda el documento activo"
 CM_FILESAVEAS, "Guarda el documento activo con un nuevo nombre."
 CM_FILEPRINT, "Imprime el documento activo"
 CM_FILEPRINTERSETUP, "Configura opciones de impresión"
 CM_FILEPRINTPREVIEW, "Muestra en páginas completas el documento"
 CM_EXIT, "Cierra impulsoApp"
 CM_EDITUNDO -1, "Operaciones de edición"
 CM_EDITUNDO, "Deshace la última acción"
 CM_EDITCUT, "Corta la selección y la coloca en el Portapapeles"
 CM_EDITCOPY, "Copia la selección y la coloca en el Portapapeles"
 CM_EDITPASTE, "Inserta el contenido del Portapapeles en el punto de inserción"
 CM_EDITDELETE, "Efectua un borrado hacia adelante o elimina la selección"
 CM_EDITCLEAR, "Borra por completo el contenido del documento"
 CM_EDITFIND -1, "Operaciones de Busqueda y Reemplazo"
 CM_EDITFIND, "Busca el texto especificado"
 CM_EDITREPLACE, "Busca el texto especificado y lo reemplaza"
 CM_EDITFINDNEXT, "Busca la siguiente ocurrencia"
 CM_CASCADECHILDREN -1, "Operaciones de Selección y Ordenamiento de ventanas"
 CM_CASCADECHILDREN, "Abre las ventanas en cascada"
 CM_TILECHILDREN, "Abre las ventanas en mosaico"
 CM_ARRANGEICONS, "Ordena las ventanas minimizadas en la parte inferior de la ventana principal"
 CM_CLOSECHILDREN, "Cierra todos los documentos"
 CM_HELPCONTENTS -1, "Accesando la ayuda en linea"
 CM_HELPCONTENTS, "Ayuda sobre el uso Impulso"
 CM_HELPUSING, "Como usar la ayuda en linea"
 CM_HELPABOUT, "Muestra los Derechos y Versión del programa"
END

// Tabla de cadens del OWL

// Editor de archivo (include\owl\editfile.rc and include\owl\editsear.rc)
STRINGTABLE
BEGIN
 IDS_CANNOTFIND, "No hay más ocurrencias de \042%s\042."
 IDS_UNABLEREAD, "Imposible leer el archivo %s desde disco"
 IDS_UNABLEWRITE, "Imposible escribir el archivo %s a disco"
 IDS_FILECHANGED, "El texto en el archivo %s  ha sido modificado.\n\nDeseas guardar los cambios he-
chos?"
 IDS_FILEFILTER, "Archivos de Texto (*.TXT)|*.TXT|Todos los Archivos (*.*)|*.*|"
END

// Doc/View (include\owl\docview.rc)
STRINGTABLE
BEGIN
 IDS_DOCMANAGERFILE, "&File"
 IDS_DOCLIST, "—Document Type—"
```

```
 IDS_VIEWLIST, "—View Type—"
 IDS_UNTITLED, "- Sin Título -"
 IDS_UNABLEOPEN, "Imposible abrir documento"
 IDS_UNABLECLOSE, "Imposible cerrar documento"
 IDS_READERROR, "Error al leer el documento"
 IDS_WRITEERROR, "Error al escribir en el documento"
 IDS_DOCCHANGED, "El documento ha sido modificado.\n\nDeseas guardar los cambios ?"
 IDS_NOTCHANGED, "El documento no ha sido modificado"
 IDS_NODOCMANAGER, "EL Administrador de archivos no esta presente"
 IDS_NOMEMORYFORVIEW, "Insuficiente memoria para verlo"
 IDS_DUPLICATEDOC, "El documento ya ha sido cargado"
END

// Impresora (include\owl\printer.rc)
STRINGTABLE
BEGIN
 IDS_PRNON, "Activa"
 IDS_PRNERRORTEMPLATE, "'%s' no fue realizado el proceso de impresión. %s."
 IDS_PRNOUTOFMEMORY, "Sin memoria"
 IDS_PRNOUTOFDISK, "Sin espacio en disco"
 IDS_PRNCANCEL, "Impresión cancelada"
 IDS_PRNMGRABORT, "Impresión cancelada desde el Adminstrador de Impresión"
 IDS_PRNGENERROR, "ha ocurrido un error durante la impresión"
 IDS_PRNERRORCAPTION, "Error de Impresión"
END

// Recursos de excepción de cadenas (include\owl\except.rc)
STRINGTABLE
BEGIN
 IDS_OWLEXCEPTION, "Excepción de ObjectWindows"
 IDS_UNHANDLEDXMSG, "Excepción no manejada"
 IDS_OKTORESUME, "OK para reanudar?"
 IDS_UNKNOWNEXCEPTION, "Excepción desconocida"
 IDS_UNKNOWNERROR, "Error desconocido"
 IDS_NOAPP, "No se encuentra el objeto de  aplicación"
 IDS_OUTOFMEMORY, "Sin memoria"
 IDS_INVALIDMODULE, "Modulo especificado invalido para la Ventana"
 IDS_INVALIDMAINWINDOW, "Ventana Principal onvalida"
 IDS_INVALIDWINDOW, "Ventana %s invalida"
 IDS_INVALIDCHILDWINDOW, "Ventana hija %s invalida"
 IDS_INVALIDCLIENTWINDOW, "Ventana cliente %s invalida"
 IDS_CLASSREGISTERFAIL, "Falla de resgitro de clase para la ventana %s"
 IDS_CHILDREGISTERFAIL, "Falla de resgitro de clase para la ventana hija %s"
 IDS_WINDOWCREATEFAIL, "Falla de creación para la Ventana %s"
 IDS_WINDOWEXECUTEFAIL, "Falla de ejecución para la Ventana %s"
 IDS_CHILDCREATEFAIL, "Falla de creación para la Ventana hija %s"
 IDS_MENUFAILURE, "Falla de creación del Menu"
 IDS_VALIDATORSYNTAX, "Error de Sintaxis en el Validator "
 IDS_PRINTERERROR, "Error desde la impresora"
 IDS_LAYOUTINCOMPLETE, "Constantes de flujo de salida incompletas definidas en la ventana %s"
 IDS_LAYOUTBADRELWIN, "Ventana %s relativa invalida"
 IDS_GDIFAILURE, "Falla en el GDI"
 IDS_GDIALLOCFAIL, "Falla al alojar el GDI"
 IDS_GDICREATEFAIL, "Falla de creación del GDI"
 IDS_GDIRESLOADFAIL, "Falla al cargar recursos del GDI"
 IDS_GDIFILEREADFAIL, "Falla en la lectura del archivo GDI"
 IDS_GDIDELETEFAIL, "Falla en la eliminación del objeto %X GDI"
```

```
 IDS_GDIDESTROYFAIL, "Falla al destruir el objeto %X del GDI"
 IDS_INVALIDDIBHANDLE, "Invalido DIB al tomar %X"
END

// Mensajes generales de la barra de estado. (include\owl\statusba.rc)
STRINGTABLE
BEGIN
 IDS_MODES, "EXT|CAPS|NUM|SCRL|OVR|REC"
 SC_SIZE, "Cambia el tamaño de la ventana"
 SC_MOVE, "Mueve la ventana a otra posición"
 SC_MINIMIZE, "Reduce la ventana a un icono"
 SC_MAXIMIZE, "Agranda la ventana a su máximo tamaño"
 SC_RESTORE, "Restaura la ventana a su anterior tamaño"
 SC_CLOSE, "Cierra la ventana"
 SC_TASKLIST, "Abre la Lista de Tareas"
 SC_NEXTWINDOW, "Cambia a la siguiente ventana"
END

// Mensajes de validación (include\owl\validate.rc)
STRINGTABLE
BEGIN
 IDS_VALPXPCONFORM, "La entrada no corresponde a la imagen:\n\042%s\042"
 IDS_VALINVALIDCHAR, "Caracter invalido en la entrada"
 IDS_VALNOTINRANGE, "Valor fuera de rango de %ld a %ld."
 IDS_VALNOTINLIST, "La entrada no es valida en la Lista"
END

// Bitmaps usados por la barra de heraamientas.  Cada bitmap  se encuentra asociado
// con un comando del menú en particular.
CM_MDIFILENEW BITMAP "new.bmp"
CM_MDIFILEOPEN BITMAP "open.bmp"
CM_FILESAVE BITMAP "save.bmp"

CM_EDITUNDO BITMAP "undo.bmp"
CM_EDITCUT BITMAP "cut.bmp"
CM_EDITCOPY BITMAP "copy.bmp"
CM_EDITPASTE BITMAP "paste.bmp"
CM_EDITFIND BITMAP "find.bmp"
CM_EDITFINDNEXT BITMAP "findnext.bmp"

CM_FILEPRINTPREVIEW  BITMAP "preview.bmp"
CM_FILEPRINT BITMAP "print.bmp"

CM_HELPCONTENTS BITMAP "help.bmp"

// Bitmaps usados por la barra de herramientas para la presentación previa.
APX_PPR_PREVIOUS BITMAP "previous.bmp"
APX_PPR_NEXT BITMAP "next.bmp"
APX_PPR_ONEUP BITMAP "preview1.bmp"
APX_PPR_TWOUP BITMAP "preview2.bmp"

// Definiciones miscelaneas de apliación

// Icono del documento MDI
IDI_DOC ICON "mdichild.ico"

// Icono de la Aplicación
```

```
IDI_MDIAPPLICATION ICON "impulso.ico"

// Caja de dialogo Acerca de ...

// Bitmaps BWCC  para caja de dialogo Acerca de ...
#define IDB_BWCC_ABOUT_ICON    1450
#define IDB_BWCC_ABOUT_ICON2   2450        // Esta definicón se encuentra definida para monitores
                                           // EGA y modos de video similares

IDB_BWCC_ABOUT_ICON  BITMAP "borabout.bmp"
IDB_BWCC_ABOUT_ICON2 BITMAP "borabout.bmp"

IDD_ABOUT DIALOG 56, 40, 199, 98
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CLASS "bordlg_gray"
CAPTION "Acerca sobre Impulso"
FONT 8, "MS Sans Serif"
BEGIN
 CONTROL "", -1, "BorShade", BSS_GROUP | BSS_CAPTION | BSS_LEFT | WS_CHILD |
                              WS_VISIBLE, 48, 6, 144, 51
 CONTROL "Version", IDC_VERSION, "BorStatic", SS_CENTER | SS_NOPREFIX | WS_CHILD |
                                              WS_VISIBLE | WS_GROUP, 51, 18, 138, 9
 CONTROL "Button", IDB_BWCC_ABOUT_ICON - 1000, "BorBtn", BBS_BITMAP | WS_CHILD |
                    WS_VISIBLE | WS_TABSTOP, 6, 9, 38, 41
 CONTROL "Tesis - Aplicación.", -1, "BorStatic", SS_CENTER | SS_NOPREFIX | WS_CHILD |
                                                 WS_VISIBLE | WS_GROUP, 51, 9, 138, 9
 CONTROL "", IDC_COPYRIGHT, "BorStatic", SS_CENTER | SS_NOPREFIX | WS_CHILD |
                                          WS_VISIBLE | WS_GROUP, 51, 27, 138, 27
 CONTROL "", IDC_DEBUG, "BorStatic", SS_RIGHT | SS_NOPREFIX | WS_CHILD | WS_VISIBLE |
                                      WS_GROUP, 131, 87, 66, 8
 CONTROL "", IDOK, "BorBtn", BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE |
                              WS_TABSTOP, 81, 66, 37, 25
END

// Caja  de dialogo para cancelación de la impresión.
IDD_ABORTDIALOG DIALOG 70, 50, 163, 65
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CLASS "bordlg"
CAPTION "Imprimiendo"
BEGIN
 CONTROL "Button", IDCANCEL, "BorBtn", BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE |
                                        WS_TABSTOP, 65, 40, 33, 21
 CTEXT "Imprimiendo", -1, 18, 8, 130, 8, SS_CENTER | SS_NOPREFIX | WS_GROUP
 CTEXT "", ID_PORT, 18, 24, 130, 8, SS_CENTER | NOT WS_VISIBLE | WS_GROUP | SS_NOPREFIX
 CTEXT "%s en %s", ID_DEVICE, 18, 16, 130, 8, SS_CENTER | SS_NOPREFIX | WS_GROUP
END

// Dialogo de entrada de la clase TInputDialog
IDD_INPUTDIALOG DIALOG 20, 24, 180, 70
STYLE WS_POPUP | WS_CAPTION | DS_SETFONT
CLASS "bordlg"
FONT 8, "Helv"
BEGIN
   LTEXT "", ID_PROMPT, 10, 8, 160, 10, SS_NOPREFIX
   CONTROL "", ID_INPUT, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP |
                                 ES_AUTOHSCROLL, 10, 20, 160, 12
   CONTROL "Button", IDOK, "BorBtn", BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE |
```

```
                                        WS_TABSTOP, 47, 42, 37, 26
    CONTROL "Button", IDCANCEL, "BorBtn", BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE |
                                            WS_TABSTOP, 93, 42, 38, 25
END

// Bitmap de secuencia deslizable Horizontal para las clases TSlider y VSlider (include\owl\slider.rc)
IDB_HSLIDERTHUMB BITMAP PRELOAD MOVEABLE DISCARDABLE
BEGIN
    '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
    '00 00 12 00 00 00 14 00 00 00 01 00 04 00 00 00'
    '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
    '00 00 10 00 00 00 00 00 00 00 00 00 C0 00 00 C0'
    '00 00 00 C0 C0 00 C0 00 00 00 C0 00 C0 00 C0 C0'
    '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
    '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
    '00 00 FF FF FF 00 BB BB 0B BB BB BB B0 BB BB 00'
    '00 00 BB B0 80 BB BB BB 08 0B BB 00 00 00 BB 08'
    'F8 0B BB B0 87 70 BB 00 00 00 B0 8F F8 80 BB 08'
    '77 77 0B 00 00 00 08 F8 88 88 00 88 88 87 70 00'
    '00 00 0F F7 77 88 00 88 77 77 70 00 00 00 0F F8'
    '88 88 00 88 88 87 70 00 00 00 0F F7 77 88 00 88'
    '77 77 70 00 00 00 0F F8 88 88 00 88 88 87 70 00'
    '00 00 0F F7 77 88 00 88 77 77 70 00 00 00 0F F8'
    '88 88 00 88 88 87 70 00 00 00 0F F7 77 88 00 88'
    '77 77 70 00 00 00 0F F8 88 88 00 88 88 87 70 00'
    '00 00 0F F7 77 88 00 88 77 77 70 00 00 00 0F F8'
    '88 88 00 88 88 87 70 00 00 00 0F F7 77 88 00 88'
    '77 77 70 00 00 00 0F F8 88 88 00 88 88 87 70 00'
    '00 00 0F F7 77 78 00 88 77 77 70 00 00 00 0F FF'
    'FF FF 00 88 88 88 80 00 00 00 B0 00 00 00 BB 00'
    '00 00 0B 00 00 00'
END

// Bitmap de secuencia deslizable vertical para las clases TSlider yd HSlider (include\owl\slider.rc)
IDB_VSLIDERTHUMB BITMAP PRELOAD MOVEABLE DISCARDABLE
BEGIN
    '42 4D 2A 01 00 00 00 00 00 00 76 00 00 00 28 00'
    '00 00 28 00 00 00 09 00 00 00 01 00 04 00 00 00'
    '00 00 B4 00 00 00 00 00 00 00 00 00 00 00 00 00'
    '00 00 10 00 00 00 00 00 00 00 00 00 C0 00 00 C0'
    '00 00 00 C0 C0 00 C0 00 00 00 C0 00 C0 00 C0 C0'
    '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
    '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
    '00 00 FF FF FF 00 B0 00 00 00 00 00 00 00 00 0B'
    'B0 00 00 00 00 00 00 00 00 00 0B 0F 88 88 88 88 88'
    '88 88 88 80 08 88 88 88 88 88 88 88 88 80 0F 77'
    '77 77 77 77 77 77 77 80 08 77 77 77 77 77 77 77'
    '77 80 0F 77 FF FF FF FF FF FF F7 80 08 77 FF FF'
    'FF FF FF FF F7 80 0F 70 00 00 00 00 00 00 77 80'
    '08 70 00 00 00 00 00 00 77 80 0F 77 77 77 77 77'
    '77 77 77 80 08 77 77 77 77 77 77 77 77 80 0F 77'
    '77 77 77 77 77 77 77 80 08 77 77 77 77 77 77 77'
    '77 80 0F FF FF FF FF FF FF FF FF F0 08 88 88 88'
    '88 88 88 88 88 80 B0 00 00 00 00 00 00 00 00 0B'
    'B0 00 00 00 00 00 00 00 00 0B'
END
```

```
// Información acerca de la Versión.
#if !defined(__DEBUG_)
// Non-Debug VERSIONINFO
1 VERSIONINFO
FILEVERSION 1, 0, 0, 0
PRODUCTVERSION 1, 0, 0, 0
FILEFLAGSMASK 0
FILEFLAGS VS_FFI_FILEFLAGSMASK
FILEOS VOS__WINDOWS16
FILETYPE VFT_APP
BEGIN
 BLOCK "StringFileInfo"
 BEGIN
  BLOCK "040904E4"
  BEGIN
   VALUE "CompanyName", "UNAM.\000"
   VALUE "FileDescription", "Impulso for Windows\000"
   VALUE "FileVersion", "1.0\000"
   VALUE "InternalName", "Impulso\000"
   VALUE "LegalCopyright", "Derechos Reservados. 1994 - 1997.\000"
   VALUE "LegalTrademarks", "Windows /231 is a trademark of Microsoft Corporation\000"
   VALUE "OriginalFilename", "Impulso.EXE\000"
   VALUE "ProductName", "Impulso\000"
   VALUE "ProductVersion", "1.0\000"
  END
 END
 BLOCK "VarFileInfo"
 BEGIN
  VALUE "Translation", 0x04e4, 0x0409
 END

END
#else

// Debug VERSIONINFO
1 VERSIONINFO LOADONCALL MOVEABLE
FILEVERSION 1, 0, 0, 0
PRODUCTVERSION 1, 0, 0, 0
FILEFLAGSMASK VS_FF_DEBUG | VS_FF_PRERELEASE | VS_FF_PATCHED |
VS_FF_PRIVATEBUILD | VS_FF_SPECIALBUILD
FILEFLAGS VS_FFI_FILEFLAGSMASK
FILEOS VOS__WINDOWS16
FILETYPE VFT_APP
BEGIN
   BLOCK "StringFileInfo"
   BEGIN
     // Tipo deLanguaje = Inglés U.S. (0x0409) y conjunto de caracteres = Windows, Multilingual(0x04e4)
     BLOCK "040904E4"                    // Empatando VarFileInfo con los valores de traslación hex.
     BEGIN
        VALUE "CompanyName", "UNAM.\000"
        VALUE "FileDescription", "Impulso for Windows\000"
        VALUE "FileVersion", "1.0\000"
        VALUE "InternalName", "Impulso\000"
        VALUE "LegalCopyright", "Copyright (r) 1996.\000"
        VALUE "LegalTrademarks", "Windows \231 is a trademark of Microsoft Corporation\000"
        VALUE "OriginalFilename", "Impulso.EXE\000"
        VALUE "ProductName", "Impulso\000"
```

```
        VALUE "ProductVersion", "1.0\000"
        VALUE "SpecialBuild", "Debug Version\000"
        VALUE "PrivateBuild", "Built by Pedro L3/4pez Urzúa.\000"
      END
    END

    BLOCK "VarFileInfo"
    BEGIN
      VALUE "Translation", 0x04e4, 0x0409      // Inglés U.S. (0x0409) & Windows Multilingual(0x04e4)
1252
    END

END
#endif
```

Archivo de definiciones IMPLSAPP.DEF

```
NAME Impulso

DESCRIPTION 'Impulso Application - Copyright © 1996.'
EXETYPE    WINDOWS
CODE       PRELOAD MOVEABLE DISCARDABLE
DATA       PRELOAD MOVEABLE MULTIPLE
HEAPSIZE   4096
STACKSIZE  8192
```

Archivo proyecto de la ayuda IMPULSO.HPJ

```
[OPTIONS]
CONTENTS=main_index
TITLE=Ayuda sobre Impulso
COMPRESS=true
WARNING=2

[FILES]
Mainhelp.rtf      ; main topics
Toolbar.rtf       ; toolbar topics
Keys.rtf          ; keyboard topics

[BITMAPS]
helpicon.bmp

[MAP]
#include <implsapp.rh>
```