INTRODUCCION

El desarrollo de este trabajo investigativo, ayuda a conocer hacia donde se dirigen las tecnologías, estructura, funcionamiento y niveles de trabajo de los sistemas operativos contemplados. Para este estudio se ha escogido las plataformas de trabajo como Windows 9X, Unix y Linux.

A la vez, el estudio de todos los ambientes gráficos de escritorio; así como sus API's de funcionamiento en cada uno de los sistemas operativos, que permitan revisar los procesos, niveles de compatibilidad, escalabilidad, y la ejecución de programas y aplicaciones.

En los casos de estudios que plantea este trabajo, se puede advertir un avance sobre los niveles de programación a través del emulador Wine, que día tras día miles de programadores ayudan a compensar este proyecto para ejecutar las funciones nativas del API de Windows, y así como tambíen los estudios sobre Máquinas Virtuales, que permiten realizar un mejor desempeño para la ejecución de muchas aplicaciones a nivel general.

Esperando que la investigación en este campo sea de mucha utilidad, para decidir si el obtener un código fuente da muchas posibilidades de establecer un buen desempeño y/o costo-beneficio para el trabajo diario de una empresa o de un usuario doméstico, frente a casas comerciales muy grandes que obtienen ganancias mayores con el desarrollo de software.

Este trabajo cubre cuatro temas importantes en el que se desarrolla todo el entorno investigativo sobre:

- **Parte 1:** este capítulo consta de una breve introducción hacia el sistema operativo Linux, su historia, técnicas de programación a nivel general dentro de Linux, la introducción a los primeras interfases gráficas con Linux, el desarrollo de C y C++ para su entorno de programación, así como algunas alternativas dentro de la programación de Linux.
- **Parte 2:** examina un estudio sobre las API's tanto de Linux como de Windows en sus diferentes versiones, los enlaces que se pueden realizar a nivel de Unix – Linux, así como el estudio en la implementación de widgets; finalizando con conclusiones directas a esta parte.
- **Parte 3:** se detalla el estudio de algunos emuladores como son: Wine y Dosemu que se integran en algunas de las distribuciones de Linux (actualmente ya vienen en todas) y los diversos lenguajes de prototipado en inicio de algún software de esta naturaleza.
- **Parte 4:** se realizan breves introducciones hacia las recompilaciones de código fuentes, la reingeniería de programación y sistemas de desarrollo de programación software multiplataforma y diversos casos de estudio con los emuladores anteriormente citados y la ejecución de aplicaciones utilizando la máquina virtual.
- **Parte 5:** se finaliza con las conclusiones y recomendaciones que se dan a este trabajo, así como sus apéndices que ayudan a investigar sobre este tema.

Se ha utilizado sintaxis especial en los siguientes casos:

- Los textos con negrilla, establece los temas, ubica a las direcciones o a la estructura sobre la que se está utilizando dentro de un directorio, así como algunos comandos de ejecución.

  Ejm: **Apis de Linux**

  ./configure

- Los textos establecidos con letra cursiva, hacen referencia a las figuras y cuadros presentados como resumen del tema expuesto, o también hacia algún significado de relevancia citado.

  Ejm: *Fig # 1. Representación de POSIX*

  SVR3: System V release 3

- Los comentarios en los programas o ejemplos de configuraciones hace referenciacia con doble slash (//) y tipo de letra Arial.

  Ejm: // Linea de Comienzo

  // Estructura de for.

CAPITULO I

INTRODUCCION

En este capítulo, se realiza un breve análisis e historia sobre el sistema operativo Linux, suficiente como para que los desarrolladores escojan técnicas de programación para realizar las aplicaciones que se requieran, aprovechando las ventajas y características que ofrece esta plataforma.

El manejo de los entornos gráficos o escritorios de Linux, ha ido superando con el paso del tiempo, en base a las nuevas versiones que presentan, el desarrollo creciente en este ámbito gracias a las principales casas desarrolladoras como KDE (K Desktop Enviroment) y GNOME (GNU Network Object Model Enviroment), que dan un buen entorno de manejo hacia el usuario.

El lenguaje de programación más utilizado es C y C++ para el desarrollo de librerías, aplicaciones y entornos de escritorios, presentan mucha facilidad y utilidad de programación, y la reutilización de código fuente escritos por otros programadores.

1.1 Breve análisis del S.O Linux

En esta era de cambios en el ambiente computacional, de una amplia oferta en sistemas operativos e interfaces gráficas y sobre todo, del costo que representa contar con un sistema operativo que interactúe con el software sin problemas, surge con fuerza inusitada: Linux

"Linux es un sistema operativo gratuito de 32 o 64 bits para redes, similar a Unix, con código abierto, optimizado para internet que puede funcionar en distintos tipos de hardware, incluso en los procesadores Intel (x68) o RISC."[1]

El sistema operativo lo forman el núcleo (kernel) mas un gran número de programas / librerías que hacen posible su utilización. Linux se distribuye bajo la GNU[2] Public License, por lo tanto el código fuente tiene que estar siempre accesible.

El sistema ha sido diseñado y programado por multitud de programadores alrededor del mundo. El núcleo del sistema sigue en continuo desarrollo bajo la coordinación de Linus Torvalds, creador de este proyecto, a principios de la década de los noventa.

Cada vez más, programas / aplicaciones están disponibles para este sistema, y la calidad de los mismos aumenta de versión a versión. La gran mayoría de los mismos vienen acompañados del código fuente y se distribuyen gratuitamente bajo los términos de la GNU Public License.

[1] Linux Máxima seguridad, Anónimo, pag 5
[2] GNU: GNU no es Unix, Referencia a licencias públicas, www.gnu.org

Los procesadores en que se puede utilizar Linux son desde un 386 y toda la familia Pentium (I – IV); también existen versiones para su utilización en otras plataformas, como Alpha, ARM, MIPS, PowerPC, Mac y SPARC. En los últimos tiempos, ciertas casas de software comercial han empezado a distribuir sus productos para Linux y la presencia del mismo en empresas aumenta rápidamente por la excelente relación calidad - precio que se consigue con Linux.

1.1.1 Reseña

Historia del Linux: Linux fue creado originalmente por Linus Torvald en la Universidad de Helsinki en Finlandia, siendo él estudiante de informática, continuando su desarrollo con la ayuda de muchos otros programadores a través de Internet.

Linux originalmente inició el desarrollo del núcleo como su proyecto favorito, inspirado por su interés en Minix, un pequeño sistema Unix desarrollado por Andrew Tannenbaum. El se propuso a crear lo que en sus propias palabras sería un "mejor Minix que el Minix".

El 5 de octubre de 1991, Linux anunció su primera versión "oficial" de Linux, versión 0.02. Desde entonces, muchos programadores han respondido a su llamada, y han ayudado a construir Linux como el sistema operativo completamente funcional que es hoy.

1.1.2 Linux

Linux es un sistema operativo diseñado por cientos de programadores de todo el planeta, aunque el principal responsable del proyecto es Linus Tovalds. Su objetivo inicial es propulsar el software de libre distribución junto con su código fuente para que pueda ser modificado por cualquier persona, dando rienda suelta a la creatividad.

El hecho de que el sistema operativo incluya su propio código fuente expande enormemente las posibilidades de este sistema. Este método también es aplicado en numerosas ocasiones a los programas que corren en el sistema, lo que hace que podamos encontrar muchos programas útiles totalmente gratuitos y con su código fuente.

Las principales funciones de este sistema operativo son:

- *Sistema multitarea.* En Linux es posible ejecutar varios programas a la vez sin necesidad de tener que parar la ejecución de cada aplicación.

- *Sistema multiusuario*. Varios usuarios pueden acceder a las aplicaciones y recursos del sistema Linux al mismo tiempo. Y, por supuesto, cada uno de ellos puede ejecutar varios programas a la vez (multitarea).

- *Shells programables*. Un shell conecta las ordenes de un usuario con el kernel de Linux (el núcleo del sistema); y al ser programables se puede modificar para

adaptarlo a tus necesidades. Por ejemplo, es muy útil para realizar procesos en segundo plano.

- *Independencia de dispositivos*. Linux admite cualquier tipo de dispositivo (módems, impresoras, etc), se añade al kernel el enlace o controlador necesario con el dispositivo, haciendo que el kernel y el enlace se fusionen. Linux posee una gran adaptabilidad y no se encuentra limitado como otros sistemas operativos.

- *Comunicaciones*. Linux es el sistema más flexible para poder conectarse a cualquier ordenador del mundo. Internet se creó y desarrolló dentro del mundo de Unix, y por lo tanto Linux tiene las mayores capacidades para navegar, ya que Unix y Linux son sistemas prácticamente idénticos. Con Linux podrá montar un servidor en su propia casa sin tener que pagar las enormes cantidades de dinero que piden otros sistemas.

En breve resumen se detalla algunas características de Linux como:

- tiene protección de la memoria entre procesos, de manera que uno de ellos no pueda colgar el sistema.
- carga de ejecutables por demanda: Linux sólo lee de disco aquellas partes de un programa que están siendo usadas actualmente.
- política de copia en escritura para la compartición de páginas entre ejecutables: esto significa que varios procesos pueden usar la misma zona de memoria para

ejecutarse. Cuando alguno intenta escribir en esa memoria, la página (4Kb de memoria) se copia a otro lugar. Esta política de copia en escritura tiene dos beneficios: aumenta la velocidad y reduce el uso de memoria.

- memoria virtual usando paginación (sin intercambio de procesos completos) a disco: una partición o un archivo en el sistema de archivos, o ambos, con la posibilidad de añadir más áreas de intercambio sobre la marcha (se sigue denominando intercambio, es en realidad un intercambio de páginas).
- la memoria se gestiona como un recurso unificado para los programas de usuario y para el caché de disco, de tal forma que toda la memoria libre puede ser usada para caché y éste puede a su vez ser reducido cuando se ejecuten grandes programas.
- librerías compartidas de carga dinámica (*DLL's*[3]) y librerías estáticas también, por supuesto.
- se realizan volcados de estado (core dumps) para posibilitar los análisis post-mortem, permitiendo el uso de depuradores sobre los programas no sólo en ejecución sino también tras abortar éstos por cualquier motivo.
- casi totalmente compatible con *POSIX*[4], *System V*[5] y *BSD*[6] a nivel fuente.
- mediante un módulo de emulación de *iBCS2*[7], casi completamente compatible con *SCO*[8], *SVR3*[9] y *SVR4*[10] a nivel binario.

[3] DLL : Dynamic Link Library (Librería de enlace dinámico)
[4] POSIX : Portable Operating System Interface (Interfaz de Portabilidad de Sistemas Operativos)
[5] System V: Kernel del sistema operativo Unix estándar, desarrollado por la compañía AT&T.
[6] BSD : Berkeley Software Distribution (Compañía de distribución de Software) con Licencia Libre
[7] iBCS2 : Intel Binary Compatibility Specification versión 2
[8] SCO : Santa Cruz Operative System
[9] SVR3: System V release 3
[10] SVR4: System V release 4

- todo el código fuente está disponible, incluyendo el núcleo completo y todos los drivers, las herramientas de desarrollo y todos los programas de usuario; además todo ello se puede distribuir libremente. Hay algunos programas comerciales que están siendo ofrecidos para Linux actualmente sin código fuente, pero todo lo que ha sido gratuito sigue siendo gratuito.
- control de tareas POSIX.
- pseudo-terminales (pty's).
- emulación de 387 en el núcleo, de tal forma que los programas no tengan que hacer su propia emulación matemática. Cualquier máquina que ejecute Linux parecerá dotada de coprocesador matemático. Por supuesto, si el ordenador ya tiene una *FPU*[11], será usada en lugar de la emulación, pudiendo incluso compilar el kernel sin la emulación matemática y conseguir un ahorro de memoria.
- soporte para muchos teclados nacionales o adaptados y es bastante fácil añadir nuevos dinámicamente.
- consolas virtuales múltiples: varias sesiones de login a través de la consola entre las que se puede cambiar con las combinaciones adecuadas de teclas (totalmente independiente del hardware de video). Se crean dinámicamente y se puede tener hasta 64.
- acceso transparente a particiones MS-DOS (o a particiones OS/2, *FAT*[12]) mediante un sistema de archivos especial: no se necesita ningún comando especial para usar la partición MS-DOS. Las particiones comprimidas de MS-DOS 6 no son accesibles en este momento, y no se espera que lo sean en el

[11] FPU: Float Point Unit (Unidad de Punto Flotante).
[12] FAT : File Access Table (Tabla de Accesos de Archivos)

futuro. El soporte para VFAT (WindowsNT, Windows 95) ha sido añadido al núcleo de desarrollo.

- un sistema de archivos especial llamado *UMSDOS*[13] que permite que Linux sea instalado en un sistema de archivos DOS.
- sistema de archivos de CD-ROM que lee todos los formatos estándar de CD-ROM.
- *TCP/IP*[14], incluyendo ftp, telnet, NFS, etc.
- Appletalk disponible en el actual núcleo de desarrollo.
- software cliente y servidor Netware disponible en los núcleos de desarrollo.

Linux no sacrifica en ningún momento la creatividad, tal y como lo hacen algunas compañías informáticas; Linux es una ventana abierta por la que es posible ir hacia un mundo donde la verdadera informática puede ser disfrutada sin límites, ni monopolios.

Linux puede adquirirse en distribuciones como RedHat, Suse, Slackware, Debían, etc. las cuales se diferencian por su método de instalación y por los paquetes (software) que viene incluído. Pero cualquier persona puede modificar un programa y venderlo según él desee, con la condición que la persona que compra ese producto puede realizar la misma acción o simplemente hacer copias para todos aquellos que lo quieran sin tener que pagar más (por lo tanto no hay que extrañarse si encuentra distribuciones comerciales). Esta licencia es la garantía que afirma la absoluta libertad de este sistema operativo.

[13] UMSDOS: Sistema de ficheros de Linux sobre una Fat.
[14] TCP/IP: Transfer Control Protocol/Internet Protocol

1.2 Análisis de Técnicas de Programación

Se expone una corta revisión de las técnicas de programación, para ilustrar sus propiedades particulares; exponer las ideas principales y sus problemáticas.

Los tipos de programación son:

- Programación no estructurada
- Programación procedimental
- Programación modular
- Programación orientada a objetos.

1.2.1 Programación no Estructurada

Comúnmente, las personas empiezan a aprender a programar escribiendo programas pequeños y sencillos, consistentes en un solo programa principal. Aquí "programa principal" se refiere a una secuencia de comandos o instrucciones que modifican datos que son a su vez globales en el transcurso de todo el programa (fig.#1).

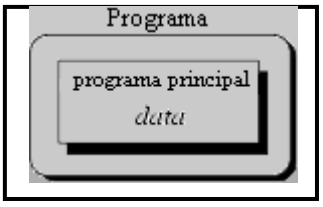


Fig # 1-1 *Programación no Estructurada.*

El programa principal opera directamente sobre datos globales

Esta técnica de programación ofrece muchas desventajas; una vez que el programa se hace suficientemente grande. Por ejemplo, si la misma secuencia de instrucciones se necesita en diferentes situaciones dentro del programa, la secuencia debe ser repetida. Esto ha conducido a la idea de extraer estas secuencias, darles un nombre y ofrecer una técnica para llamarlas y regresar desde estos procedimientos.

1.2.2 Programación Procedimental

Con la programación procedimental, se puede combinar las secuencias de instrucciones repetibles en un solo lugar. Una llamada de procedimiento se utiliza para invocar al procedimiento, después de que la secuencia es procesada, el flujo de control procede exactamente después de la posición donde la llamada fue hecha (Fig.# 2).

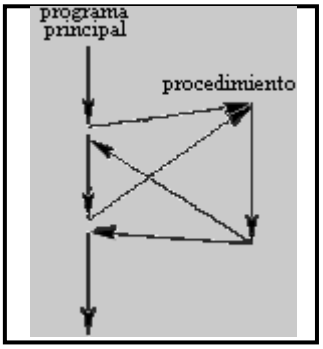


Fig # 1-2. *Ejecución de procedimientos.*

Después del procesamiento, el flujo de controles procede donde la llamada fue hecha.

Al introducir parámetros, así como procedimientos de procedimientos (subprocedimientos) los programas ahora pueden ser escritos en forma más estructurada y libres de errores. Por ejemplo; si un procedimiento ya es correcto, cada vez que es

usado produce resultados correctos. Por consecuencia; en caso de errores, se puede reducir la búsqueda a aquellos lugares que todavía no han sido revisados.

De este modo, un programa puede ser visto como una secuencia de llamadas a procedimientos. El programa principal es responsable de pasar los datos a las llamadas individuales, los datos son procesados por los procedimientos y una vez que el programa ha terminado, los datos resultantes son presentados.

Así, el flujo de datos puede ser ilustrado como una gráfica jerárquica, un árbol, para un programa sin subprocedimientos (fig. # 3).

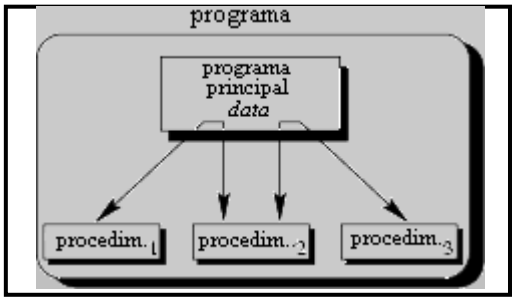


Fig # 1-3. *Programación Procedimental.*

El programa principal coordina las llamadas a procedimientos

y pasa los datos apropiados en forma de parámetros.

Para resumir, se tiene un programa único que se divide en pequeñas piezas llamadas procedimientos. Para posibilitar el uso de procedimientos generales o grupos de procedimientos también en otros programas aquéllos, deben estar disponibles en forma separada.

1.2.3 Programación Modular

En la programación modular, los procedimientos con una funcionalidad común son agrupados en módulos separados. Un programa por consiguiente, ya no consiste solamente de una sección; ahora está dividido en varias secciones más pequeñas que interactúan a través de llamadas a procedimientos y que integran el programa en su totalidad. (Fig. # 4).

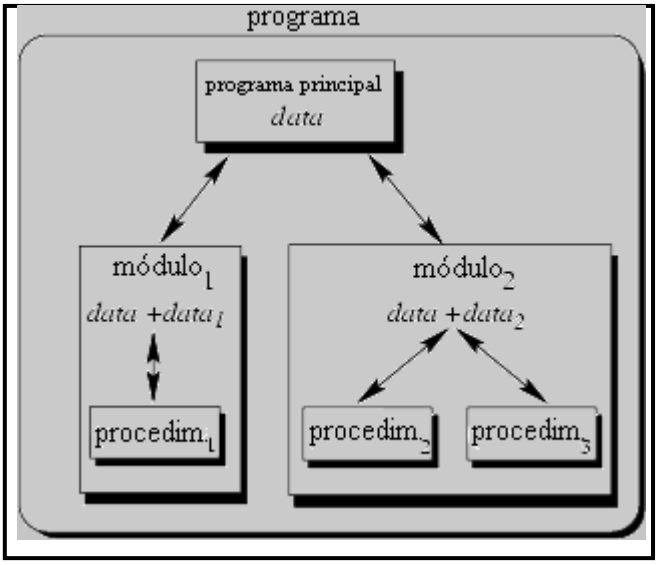


Fig # 1- 4. *Programación Modular.*

El programa principal coordina las llamadas a procedimientos en módulos separados y pasa los datos apropiados en forma de parámetros.

Cada módulo puede contener sus propios datos. Esto permite que cada módulo maneje un estado interno que es modificado por las llamadas a procedimientos de ese módulo. Sin embargo; solamente hay un estado por módulo y cada módulo existe cuando más una vez en todo el programa.

1.2.4 Programación Orientada a Objetos

La programación orientada a objetos resuelve algunos de los problemas que se acaban de mencionar. En contraste con las otras técnicas, ahora tenemos una telaraña de objetos interactuantes, cada uno de los cuáles manteniendo su propio estado (Fig. # 5).

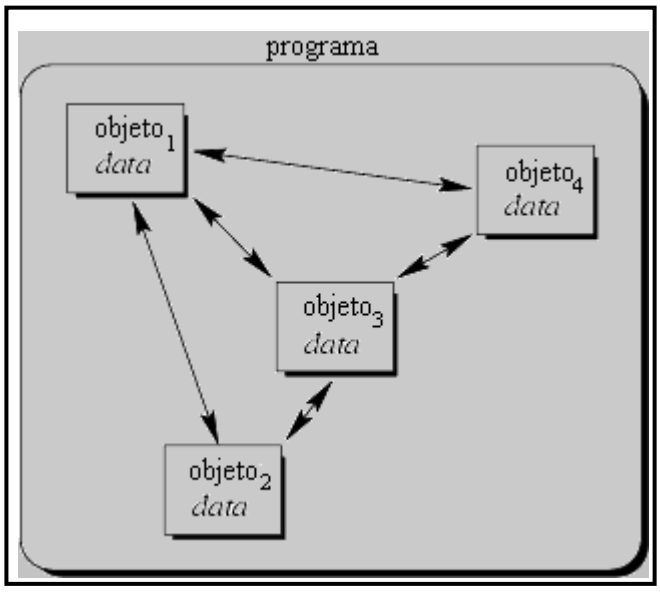


Fig # 1-5. *Programación Orientada a Objetos.*

Los objetos del programa interactúan mandando mensajes unos a otros.

Considera el ejemplo de listas múltiples nuevamente. El problema aquí con la programación modular es, que se debe crear y destruir en forma explícita los manejadores de lista; entonces se usa los procedimientos del módulo para modificar cada uno de los manejadores.

En contraste con eso, en la programación orientada a objetos se debería tener tantos objetos-lista como sea necesario. En lugar de llamar un procedimiento al que se debe proveer el manejador de lista correcto, enviaría un mensaje directamente al objeto-lista en cuestión. En términos generales, cada objeto implementa su propio módulo, permitiendo por ejemplo que coexistan muchas listas.

Cada objeto es responsable de inicializarse y destruirse en forma correcta. Por consiguiente, ya no existe la necesidad de llamar explícitamente al procedimiento de creación o de terminación.

1.3 Análisis y Técnicas de Programación para Linux

Escribir el código fuente[15] y compilar aparte, utilizando un entorno de desarrollo integrado IDE[16] permite editar y compilar desde un mismo programa; incluso la depuración integrada del mismo, sirve para realizar los proyectos que caracterizan a Linux.

1.3.1 Las librerías

El sistema operativo Linux contiene una serie de librerías básicas y también muchas adicionales, que proveen a las aplicaciones que se programe una enorme cantidad de funciones. Estas librerías son conceptualmente iguales a los DLL de Windows; y bajo

[15] Código Fuente o también llamado Source
[16] IDE: Integrated Development Environment

Linux se llaman 'Shared Libraries' (Librerías Compartidas); porque las pueden utilizar varios procesos (programas ejecutándose) al mismo tiempo.

Esto trae ventajas e inconvenientes, ya que el tamaño del binario de nuestros programas se verá reducido, porque utilizará llamadas a librerías que contienen las funciones que desea llamar. Si utiliza una librería que no está presente en todos los sistemas, el programa no funcionará al intentar ejecutarlo; el cual dirá que falta una librería. Para resolver este problema podemos hacer dos cosas:

- Conseguir la librería o una nueva versión de la misma e instalarla en el sistema.
- Conseguir una versión compilada estáticamente (static) del programa.

Compilada estáticamente, significa que al momento de compilar el programa el compilador agrega al código binario las librerías compartidas que utiliza, para que sea independiente de las mismas. Esto trae como ventaja que el binario se pueda ejecutar en cualquier sistema, ya que no necesita librerías compartidas. Y como desventaja, el tamaño va a ser grande.

Es práctica común, que en los *Home Site*[17] de los programas para Linux se encuentren versiones static y shared (compartidas), siendo shared las comunes, y también el código fuente de el/los programa/s en cuestión.

[17] Home Site, se refiere al directorio raiz (root – home) de Linux.

La mayoría de los programas de Linux proveen al usuario que los utiliza de una licencia llamada *GPL*[18], que entre otras cosas significa que el código fuente del programa lo puede utilizar cualquiera, para sus propios programas; o para modificar una versión del programa original, pero que esas partes mantienen el copyright del autor original, y se les debe dar crédito a ellos por las partes que se utilice en los programas. Así es como creció Linux.

1.4 Introducción a los manejos de interfaces gráficas en Linux

Con la aparición de nuevas versiones de Linux, se incorpora interfaces de usuario estándar y sistemas de ventanas para aplicaciones basadas en caracteres y gráficas. La interfaz gráfica de usuario se denomina *Open Look*[19], que ofrece una forma consistente, efectiva y eficaz de interactuar con las aplicaciones.

Proporciona varias posibilidades de gestión y operación de ventanas, utilización de barras de desplazamiento, claves de pulsar, pulsadores, menús, ventanas emergentes y facilidades de ayuda.

Se proporciona una caja de herramientas gráficas que puede utilizarse para construir aplicaciones Open Look. Otra interfaz gráfica de usuario importante, *Motif*[20], desarrollada originalmente por la Open Software Foundation, ha llegado a ser mucho más popular que Open Look en los últimos años.

[18] GPL: General Public License (Licencia Pública General)
[19] Open Look: Interfaz Gráfica desarrollada por Sun Microsistems, Xerox y ATT, en búsqueda del primer estándar de aplicaciones bajo el procesador sparc.
[20] Motif: Es un interfaz gráfico para PC, más ampliamanete difundido por los sistemas Unix.

1.4.1 Gestores de Ventanas

El administrador de ventanas en el sistema XWindow es una aplicación cliente, que suministra la gestión de ventanas básicas y las funciones de manipulación que utiliza al interactuar con el sistema. Esto incluye la disposición básica de las ventanas, bordes, apariencia de menús, creación y eliminación de ventanas, movimiento de ventanas, manejo de teclado y trazados de colores, además de la iconización de ventanas.

Todas estas funciones hacen que el administrador de ventanas sea el principal determinante de la apariencia y funcionamiento general del sistema.

Al igual que el sistema Unix reforzó la innovación de interfaces de usuario orientadas a caracteres, o *"shell"*, exportando las funciones de interfaz de usuario fuera del sistema operativo a un módulo separado, se permite la existencia de muchas otras versiones como el Shell C, Bourne shell, Korn shell, etc.; con lo que el sistema XWindow pone su propia interfaz con el usuario en un módulo de software separado llamado administrador de ventanas.

Y al igual que la naturaleza modular del shell conduce a shells alternativos; también se han desarrollado muchos administradores de ventanas alternativos. El sistema XWindow no dicta un "aspecto y funcionamiento" específico; por ejemplo, tal como una *GUI*[21] de PC pueden tener dos apariencias y estilos de operación muy diferentes. Podrán diferir en la forma en que los menús y sus acciones son representadas, en la

[21] GUI: Graphical User Interfase (Interfaz Gráfica de Usuario)

manera en que nuestra aplicación convierte una ventana en un icono y en otras características fundamentales. Aunque esta flexibilidad tiene un valor, las inconsistencias resultantes podrán anular las ventajas potenciales de las interfaces gráficas.

Para evitar este problema, se han desarrollado diversos productos que pretenden suministrar una interfaz de usuario consistente tanto para el sistema Unix completo como para aplicaciones de diferentes fabricantes. Los administradores de ventanas con mayor amplitud de uso y distribución son el Administrador de Ventanas Motif, *mwm*[22], de Open Look, *olwm*[23], de Sun Microsystems, juntos con otros relativos como olvwm, el cual maneja ventana en una "pantalla virtual" mucho mayor que la ventana real que se encuentra delante el usuario.

La mayoría de las aplicaciones del sistema XWindow son construidas a partir de objetos de propósitos generales de software reutilizable llamados "widgets". Las bibliotecas de esos objetos son conocidas como *toolkits*[24]. Los toolkits más conocidos se guían por las convenciones de los interfaces de usuario Motif / *CDE*[25] u Open Look, mwm y olwm fueron escritos para trabajar de una manera consistente con los widgets de los kits de herramientas Motif y Open Look.

[22] mwm: Motif Window Manager
[23] Olwm: Open Look Window Manager
[24] Toolkits: kit de herramientas
[25] CDE: Common Desktop Environment.

1.4.2 Funciones del Gestor de Ventanas

Una de las principales funciones del gestor de ventanas es arbitrar la compartición del espacio de la pantalla entre las ventanas activas simultáneamente en diferentes aplicaciones. Por esta razón, el administrador de ventanas controla la interfaz de usuario para mover, reajustar y dar forma a las ventanas de la aplicación.

Muchos administradores de ventanas utilizan la metáfora de solapar hojas de papel sobre un escritorio para ajustar el orden de apilamiento de las ventanas solapadas, dando al usuario la posibilidad de mover las ventanas hacia "atrás", para que se aposenten debajo de otras sobre la superficie del escritorio, o en el "frente" metafóricamente encima de todas las demás ventanas y papeles, sin oscurecer al usuario. Las ventanas de aplicaciones no usadas temporalmente pueden ser iconificadas en una pequeña ventana de dibujo llamado icono, de nuevo bajo la dirección de la administración de ventanas.

Para poder controlar estas funciones específicas de ventanas, la mayoría de administradores muestran un marco alrededor de cada ventana de aplicación, con controles de desplazamiento de ratón, tales como las esquinas para reajustar el tamaño de una ventana.

El resto de juego de funciones específicas de ventana están normalmente disponibles desde un menú, que aparece cuando se pulsa el botón del ratón apropiado en la barra de títulos de la ventana de aplicación.

Además de los marcos que coloca alrededor de cada ventana aplicación, el administrador de ventanas también controla la ventana *"raíz"*, el origen sobre el cual se mostrarán las ventanas de aplicación. Cuando pulsamos el botón invocador de menús del ratón sobre la ventana raíz, se obtiene el menú de funciones del administrador de ventanas que pertenecen por completo al servidor de sistema XWindow y no solamente a la ventana de alguna aplicación específica.

1.4.3 Sistema XWindows

"El sistema X Windows es un potente entorno operativo gráfico que da soporte a muchas aplicaciones en red. Ha sido desarrollado por el Instituto Tecnológico Massachussets (MIT) y es de libre distribición"[26].

La interfaz de usuario es la parte del sistema que define cómo se interactúa con él; cómo se introduce las órdenes y otros tipos de información y cómo muestra el sistema las sugerencias y la información. Incluye tanto la apariencia característica del sistema como su operación. Para la mayoría de los usuarios, la interfaz primaria al sistema Unix ha sido la interfaz de línea de órdenes suministrada por el shell. Las interfaces gráficas de usuario hacen que el trabajo con el sistema Unix sea más fácil, más efectivo y más divertido.

Ahora, las interfaces gráficas son de uso común en los PC y las GUI del sistema comparten con ellas muchas utilidades. No obstante, las interfaces gráficas de usuario

[26] Linux 4. Edición, Jack Tackett, jr. Steven Burnett, pág. 517

tienen algunas características especiales que se ajustan a las necesidades particulares de las aplicaciones.

Específicamente, los entornos gráficos deben soportar aplicaciones en red, deben permitir que las aplicaciones sean independientes de monitores específicos y de hardware de terminales, deben permitir que las aplicaciones gráficas se puedan mover fácilmente a través de la gran variedad de hardware.

El sistema XWindow incorpora todas las ventajas de interfaz de usuario de sus contemporáneos (Apple Macintosh, Microsoft Windows.). Al mismo tiempo, continúa la filosofía de Unix de ser modular y por ello más fácil de invocar; porque los reemplazos experimentales para pequeñas herramientas modulares son más fáciles de construir que los reemplazos para un complejo conglomerado de sistema operativo, sistema de ventanas y un administrador de interfaz de usuario igual al de Apple Macintosh o Windows 95.

Los principales conceptos en los que se basa el sistema XWindow incluyen un modelo cliente - servidor para la interacción entre las aplicaciones y los dispositivos de terminales, un protocolo de red, varias herramientas de software que podrán ser usadas para crear aplicaciones basadas en XWindow y una colección de aplicaciones útiles que suministran utilidades básicas para nuestras aplicaciones.

El sistema XWindow es un sistema de ventanas en red. Lo que quiere decir, que un servidor X podrá suministrar una interfaz de usuario no sólo a todos aquellos procesos

clientes que estén corriendo sobre la misma computadora o estación de trabajo como el propio servidor X; sino que también a aquellos programas que se estén ejecutando en otras computadoras conectadas a la misma red; incluso una red muy amplia, tal como la red global Internet.

Bajo Linux – Unix, el cliente se conecta al inicio con el servidor X designado por el valor de la variable de entorno *$DISPLAY* o por el argumento que sigue a la opción display en su línea de órdenes. El valor se inicia con un identificador de punto final, como un nombre *DNS*[27] o una dirección numérica IP.

Tres identificadores reservados se refieren a los procesos del servidor X que corren bajo el mismo sistema que el cliente. El intensificador de punto final se sigue por dos puntos (:) y un número de display. Este es un pequeño número que identifica un servidor X específico en la dirección de punto final, habitualmente un 0 sobre una plataforma que soporta únicamente un servidor X a la vez.

Un servidor X permite a varios procesos cliente compartir acceso al hardware que les proporciona una interfaz con el usuario. Este hardware; área de pantalla, las teclas, posición del puntero y botones necesitan ser compartidos de alguna manera por los procesos de los clientes. En algunas excepciones, un servidor X comparte sus recursos entre procesos de clientes, llamado administrador de ventanas.

La compartición de los recursos, se hacen bajo el control del usuario. Este control requiere una interfaz de usuario interactiva, la mayoría de los administradores de

[27] DNS: Domain Name Server

ventanas crean y controlan un macro alrededor de cada ventana cliente para permitir al usuario identificar, mover, reajustar, y calibrar hacia arriba o hacia abajo las ventanas aplicación en la pantalla.

La mayoría de los administradores de ventanas también suministran uno o más menús, habitualmente invocados desde el marco de la ventana para funciones específicas, o desde el fondo, o ventana raíz para otras funciones.

El cliente administrador de ventanas es especial en algunos aspectos; por ejemplo, sólo un administrador de ventanas podrá conectarse a un servidor X en un momento dado. Pero en muchos aspectos no es más que otro cliente.

Por ejemplo, no se requiere que el administrador de ventanas corra en la misma máquina que el servidor que controla y para servidores X corriendo sobre hardware dedicado "terminales X" el administrador de ventanas al igual que todos los clientes, debe ser ejecutado desde sistemas Linux - Unix en la red.

1.4.4 El Modelo Cliente- Servidor

"X Windows es un sistema Cliente/servidor controlado por dos piezas de software, una ejecutándose en el cliente y otra en el servidor. Las piezas cliente y servidor de este rompecabezas pueden estar en sistemas distintos o, como en el caso de la mayoría de computadoras personales, ambos pueden residir en la misma máquina"[28].

[28] Linux 4. Edición, Jack Tackett, jr. Steven Burnett, pág. 519

Un concepto XWindow fundamental es la separación entre las aplicaciones y el software que maneja las entradas y salidas de los terminales.

Todas las interacciones con los dispositivos terminales mostrando información sobre una pantalla, coleccionando pulsaciones de teclado o de ratón son manejados por un programa dedicado que es el completo responsable del control del terminal. Las aplicaciones (clientes) envían al servidor la información que deberá ser mostrada y el servidor envía información de aplicaciones sobre las entradas del usuario.

El separar las aplicaciones (cliente) del software que maneja el display (el servidor) significa que solamente debe conocer los detalles del software terminal o cómo debe controlarlo. El servidor "oculta" las características específicas del hardware de terminales de las aplicaciones; esto hace que sea más fácil desarrollar aplicaciones y trasladar aplicaciones del sistema X Window a nuevos terminales.

Por ejemplo, las instrucciones para dibujar una línea difieren en dos terminales diferentes. Sí una aplicación se comunica directamente con el terminal, requerirán de diferentes versiones de la misma aplicación.

Sin embargo, si las instrucciones específicas del hardware son manejadas por servidores, una aplicación podrá enviar la misma instrucción al servidor asociado a cada terminal, como resultado la misma aplicación podrá ser usada por diferentes dispositivos de terminales.

1.4.5 El Protocolo X

El protocolo X es el lenguaje estándar utilizado por aplicaciones clientes para enviar instrucciones a los servidores X y los servidores para enviar información a los clientes. En el sistema XWindow, los clientes y servidores se comunican a través del protocolo X.

El protocolo X está diseñado para trabajar sobre una red o un solo procesador. Los mensajes que envían entre el cliente y el servidor son los mismos cuando el cliente y el servidor están sobre la misma estación de trabajo o en diferentes máquinas.

Este uso de un protocolo de red como interfaz única y estándar entre cliente y servidor significa que las aplicaciones del sistema XWindow, inicialmente desarrolladas para correr sobre una estación de trabajo que tenga su propio display conectado, podrán ahora de manera automática correr sobre una red.

1.4.6 Uso de Recursos del Sistema X Window.

En algunas excepciones, el comportamiento y la apariencia de las aplicaciones del sistema XWindow están controlados por la jerarquía de variables estructuradas llamadas recursos. Los valores de las variables de recursos son guardadas en una base de datos en el proceso del servidor X; esto permite a cualquier aplicación cliente que se conecte a nuestro servidor de sistema XWindow obtener sus valores, independientemente de la parte de la red donde esté ejecutándose.

El guión del shell para empezar nuestra sesión X, habitualmente .xsesion o .xinitrc, incluye una orden como ésta:

Xrdb –load .xdefaults.

Esta orden lee el contenido de un archivo de recursos, como .Xdefaults en este ejemplo, hacia la base de datos de recursos en nuestro servidor X, éste lee esos valores y se personaliza de acuerdo a ellos. La mayoría de las aplicaciones leen la base de datos de recursos una vez, y después mantiene una copia privada de los valores.

Por lo tanto, es posible empezar con una copia de una aplicación para después cambiar el contenido de la base de datos de recursos y empezar con otra copia de la aplicación con una línea de órdenes idéntica y hacer que se comporte de manera diferente porque los valores de algunas variables de recursos han cambiado.

El archivo del cual se leen estos valores en la base de datos de recursos tiene un formato muy específico. Consiste en líneas separadas por caracteres de nueva línea, cada línea asigna un valor a un recurso individual o a una clase de recursos, perteneciente a un objeto específico dentro de alguna aplicación o a una clase de dichos objetos. Si una simple asignación de un valor a un recurso es demasiado larga para ajustare a una línea del archivo, se podrá evitar el uso de nuevas líneas mediante una barra inclinada a la izquierda (\e) al final de una línea física, para fusionar dos o más línea física en una nueva "línea lógica".

Y si el carácter literal de línea nueva necesita ser incorporado al valor asignado a la variable de recursos, se escribirá como "\en". El archivo de recursos podrá contener incluso comentarios, las líneas que empiecen con el signo de admiración en la primera columna son ignoradas por xrdb pero se mantienen en el archivo para que puedan ser leídas por nosotros.

1.4.7 Color

Los colores son uno de los rasgos más fáciles de personalizar los recursos. Cada objeto tiene al menos dos recursos de color, background y foreground. Los *widgets*[29] que aparecen dentro de otros widgets a menudo tienen un borde opcional alrededor de ellos, especificado con recursos borderWidth y borderColor.

Los colores podrán ser específicos por el nombre, utilizando los nombres de la base de datos de colores como indica showrgb, o con tres números de dígitos dobles hexadecimales en formato *#RRGGBB*[30]. El juego de colores disponibles en una pantalla dada se conoce, en la terminología sistema X Window, como el visual de la pantalla.

Los visuales comunes incluyen monocromo (blanco y negro), grayscale, true color y mapped color, también llamado pseudo-color. Pseudo-color es el que se encuentra con mayor frecuencia en el visual del sistema X Window.

[29] Widget: Formas de dibujo u objetos basados en capas.
[30] RGB: Se refiere a los colores primarios (Rojo, Verde, Azul)

Las pantallas pseudo-color pueden mostrar hasta 256 colores diferentes a la vez, de 16 millones de color verdadero especificables. Los valores de un byte en la memoria de vídeo de la computadora son asignados por un mapa de colores con hasta 256 celdas de color. Algunos administradores de ventanas como (olwm) y herramientas de personalización de plataformas (xset) tienen facilidades para la manipulación de mapas de colores.

En la actualidad las bibliotecas de sistema X Window utilizan un modelo de computadora del sistema visual humano para decidir si una combinación de intensidades dadas de rojo, verde y azul claro deberían ser representadas por blanco y negro para el ojo humano. La necesidad de personalizar los colores se asocia frecuentemente con las aplicaciones que realizan asignaciones de colores por omisión sin considerar la utilización de la aplicación en una pantalla monocroma.

1.4.8 Mapas de Bits y de Pixels.

Los mapas de bits y los mapas de pixels son formatos de representación de imagen en el sistema X Window. Los mapas de pixels podrán tener cualquier profundidad correspondiente al número de bits por pixel en algún visual del sistema X Window. Los mapas de bits son mapas de pixels monocromos, en cualquier lugar donde se espere un mapa de pixels, podrá utilizarse un mapa de bits, pero no al contrario. Bajo el sistema operativo sistema Linux - Unix los mapas de bits y de pixels del sistema X Window se guardan como archivos ordinarios.

Debido a su obvia aplicación en el almacenamiento de imágenes, los mapas de bits son utilizados en sistemas X Window de diferentes maneras, definiendo incluso las texturas del fondo de las ventanas de los objetos, para las formas de las máscaras e imágenes del puntero del cursor, para iconos y para *glyphs*[31].

Existen fuentes que contienen iconos, cursores, glyphs, incluso imágenes de algunos juegos como ajedrez y otros. Estos recursos han sido suministrados de manera que las ventanas objetos en pantallas monocroma pudieran tener fondos que contrasten con caracteres y gráficos blancos y negros, y bordes que fuesen visibles sobre fondos blancos y negros de otras ventanas. El mapa de bits habitualmente utilizado para estos propósitos está **/usr/include/X11/bitmaps/gray**.

No obstante, en teoría se puede utilizar cualquier otro mapa de bits. Esto significa que muchos widgets de sistema X Window podrán ser mostrados con fondos o dibujos de patrones diversos.

1.4.9 Fuentes

Una fuente es una colección de mapas de bits numerados, y no tienen por qué ser usados únicamente por los caracteres de un lenguaje escrito. Las fuentes se utilizan en el sistema XWindow para muchas colecciones de mapas de bits.

[31] Glyphs: elementos pictóricos de gráficos compuestos

1.4.10 El Teclado y el Ratón

El usuario se comunica con las aplicaciones del sistema XWindow y con otros clientes, como el administrador de ventanas, por medio de un teclado y un puntero, que puede ser un ratón o un joystick. Los movimientos del ratón tienen dos parámetros personalizables: la aceleración y el umbral.

La aceleración y el umbral del ratón son parámetros del servidor X, pueden personalizarse mediante la utilidad xset. Esto permite configurar el ratón para un uso de alineación precisa cuando se mueve despacio o para que salte más pixels si preferimos un movimiento más rápido.

1.4.11 La Biblioteca X

El sistema XWindow suministra un juego estándar de rutinas en lenguaje C que los programadores podrán utilizar para programar funciones gráficas y que de manera automática producen el protocolo X correspondiente. Estas rutinas se conocen como rutinas de la biblioteca X, o *Xlib*. Xlib suministra una interfaz de programador estándar para el sistema XWindow.

1.4.12 Conjunto de Herramientas (Toolkits)

Xlib suministra funciones relativamente de bajo nivel. Gestiona los elementos gráficos básicos, como el dibujar una línea, rellenar una región, etc. Para simplificar en un

futuro el desarrollo de una aplicación, las rutinas de mayor nivel han sido desarrolladas para crear elementos más complejos; por ejemplo, ventanas, menús o barras de desplazamiento. Los elementos de nivel superior se denominan widgets. Un toolkit se denomina a menudo widget set. Los widgets típicos incluyen barras de desplazamiento, botones, formularios y componentes similares.

El paquete de distribución del sistema X Window incluye una biblioteca llamada Toolkit Intrinsic (libXt, con funciones cuyos nombres empiezan con el prefijo *"Xt"*). La biblioteca de herramienta intrínseca es una fundación sobre la cual distintos fabricantes podrán construir kits de herramientas que soporten sus interfaces gráficas de usuario.

Hay dos grupos de fabricantes que han desarrollado kits de herramientas intrínsecos.

- Olit: Un kit de herramientas desarrollado por Sun, AT&T y Novell, soporta su GUI estándar Open Look.
- La Open Software Foundation, representada por distintos fabricantes incluyendo HP, IBM, DEC, y NCR, distribuye un kit de herramientas intrínseco llamado Motif, el cual soporta la GUI Common Desktop Environment (CDE).

1.4.13 GTK

Gtk (Gimp ToolKit) es el producto del proyecto GNU en lo que a librería gráfica se refiere. En el origen, Gtk fue concebida para el desarrollo de Gimp (GNU Image

Manipulation Program), pero luego se independizó de éste. Gtk está escrita en C y para ser usada desde programas en C.

1.4.14 QT

Qt es la primera librería gráfica desarrollada en programación objeto de gran calidad. Es la principal competidora de Gtk y ha sido usada para el desarrollo de KDE (K Desktop Environment), un entorno integrado de aplicaciones y gestión de ventanas disponible para cualquier Linux-Unix y que supera en muchos aspectos al interfaz gráfico de Windows.

1.5 C, C++ Y JAVA Sobre Linux

Los desarrolladores de productos muestran un interés mayoritario por programar bajo la plataforma Java en beneficio de los sistemas compatibles Unix – Linux y de modelos de aplicaciones de *Gnome* y *Kde*. Hay casos en que las aplicaciones nativas son preferibles, sobre todo cuando hay suficientes clientes en una plataforma, como para que sea rentable crear una versión propia.

Java tiene otras ventajas aparte de ser multiplataforma, se destacará simplemente dos:

- El *API* (Aplication Programming Interface) sus características como lenguaje rebajan considerablemente los costes en el ciclo de vida del software, por su sencillez, fiabilidad y orientación a objetos.

- Java esta basado en el carácter interpretado donde su costo en rendimiento es la seguridad.

La alternativa a los compiladores que traducen a código máquina nativo tienen todo el tiempo para optimizar el código generado. Esto es el caso de Java, C y C++ del cual requieren de un runtime muy complejo que realice recolección de basura y la sincronización. Como recolector de basura es frecuente utilizar una implementación muy eficiente que fue creada para sustituir en los programas de C y C++ las invocaciones a malloc.

En Java, si el programa compilado es una aplicación dinámica que carga en tiempo de ejecución nuevas clases de un repositorio para que funcione el runtime deberá integrar también el cargador y un intérprete de bytecodes e incluso para obtener un rendimiento óptimo.

La calidad de los compiladores está en continuo crecimiento y se espera que continúe así durante mucho tiempo. A ello contribuirán tecnologías de diferentes casas comerciales mediante un análisis de perfilado, de que código compila y cual no, con que optimizaciones, etc.

Algunas personas ven posibilidades excepcionales en la apenas investigada compilación dinámica, al poder utilizar información crítica que en cambio no está disponible para un compilador estático como del lenguaje C. Incluso se especula que algunos programas en Java podrían ir más rápidos que sus equivalentes en C++.

Los grandes entornos de desarrollo entran precisamente dentro de ese grupo de aplicaciones que por sus necesidades de prestaciones no deberían implementarse en Java sino en código nativo.

1.6 Reseña de otras alternativas para programación en Linux

Linux fue inicialmente un sistema desarrollado por y para programadores. Por eso cuenta con una gran cantidad de herramientas de programación. Prácticamente cualquier lenguaje de programación conocido puede usarse en Linux.

1.6.1 Programación en C

El lenguaje C es el estándar a la hora de programar aplicaciones relativamente complejas. Casi todo el núcleo de Linux está escrito en este lenguaje, y como se dijo anteriormente su evolución pareja al desarrollo de Unix lo convierten en ideal para programar en este entorno.

El compilador de C usado en la totalidad de las instalaciones de Linux es el de GNU. El GCC es un compilador rápido y eficiente, totalmente compatible con el estándar ANSI C, pero que implementa algunas características propias.

La sintaxis habitual para invocar al compilador es similar a la siguiente:

gcc fich1.c fich2.c -o fich -lm -laaa

donde fich1.c y fich2.c son los nombres de los ficheros en C donde reside el código fuente; fich es el nombre del fichero ejecutable que se pretende generar, y -lm -laaa especifican las librerías con las que se van enlazar los ficheros (en este caso las librerías libm.so y libaaa.so).

La programación en C se completa con algunas otras herramientas como make, autoconf, libtool, automake,...

La utilidad make facilita la gestión de programas con gran número de ficheros, ya que ayuda de una manera eficiente las diferentes dependencias que existen entre los ficheros y permite regenerar sólo los ficheros que necesitan en cada momento.

Por otro lado, autoconf permite generar código muy portable que se adapte por sí mismo a diferentes plataformas y arquitecturas, logrando que el mismo código fuente pueda ser compilado sin problemas en diferentes sistemas.

Existen multitud de librerías disponibles para ser enlazadas con programas escritos en C, ahorrando así tiempo y esfuerzo. Estas librerías cubren campos como: programación gráfica, resolución de ecuaciones, acceso a internet, entornos de ventanas, tratamiento de imágenes, bases de datos, criptografía, cálculo numérico, reutilización de componentes, comunicación entre aplicaciones, hebras y procesos concurrentes, acceso a hardware, impresión, etc.

A la hora de programar interfaces gráficas para el usuario, se puede elegir entre varias librerías. Algunas son:

- Motif. Es el estándar en otros sistemas operativos Unix, aunque en Linux no se usa mucho porque es comercial.
- GTK. Una librería de desarrollo abierto que ha ganando popularidad. En ella se basan el programa GIMP, y el entorno GNOME, entre otros.
- Qt. Desarrollada por Troll Tech, es la base para el entorno de escritorio KDE.

También existen entornos integrados de desarrollo (IDEs) (por ejemplo, KDevelop) y algunos programas de creación visual de aplicaciones (RADs[32]), como Glade.

1.6.2 Lenguajes de scripts

En oposición a los lenguajes compilados (como C) están los lenguajes interpretados. Normalmente, un intérprete lee un fichero y ejecuta línea a línea su contenido, estos ficheros son de texto plano, facilitando su creación y depuración.

En Unix, cualquier programa interpretado puede ser invocado directamente si comienza por los caracteres *'#!'* seguidos del nombre del intérprete. Por ejemplo, un fichero cuya primera línea sea

#!/bin/sh

[32] RAD: Rapid Application Development (Desarrollo rápido de Aplicaciones)

y tenga permisos de ejecución se puede invocar por su nombre, y el sistema se encarga de llamar a **/bin/sh** para que lo ejecute.

Algunos de los lenguajes de scripts más usados en Linux son:

- *Shell*. La shell bash también es programable. Un programa puede incluir las órdenes habituales en la línea de comandos, además de instrucciones propias de programación para control de flujo, asignación de variables, etc. Es muy conveniente para un administrador o usuario conocer el lenguaje de shell. Los programas interpretados por la shell se ejecutan con /bin/sh.

- *Perl*. "(Practical Extraction And Report Language) es un lenguaje para escribir programas, que tiene todas las capacidades de GAWK[33] pero con más características. Originalmente, Perl se diseñó para operar en archivos, como lo hace GAWK, generando reportes y manejando archivos muy grandes"[34]. Conjuga lo mejor de la programación de shell y el lenguaje C.

- *Tcl / tk*. Un lenguaje sencillo y portable que cuenta como principales ventajas la posibilidad de empleo de una interfaz gráfica (Tk) y su facilidad para extenderse mediante rutinas escritas en C.

[33] GAWK es la versión GNU de la utilidad de UNIX
[34] Fundamentos de programación en Linux, Petersen Rcichard, pág. 188

Es muy conveniente realizar algunos programas simples interpretados por la shell, ya que pueden facilitar mucho la tarea de administrar el sistema, y logran evitar la realización de tareas repetitivas.

Un programa simple en bash puede tener la forma siguiente:

```
#!/bin/sh                    // llama al proceso bash
if [ "$1" = -h ]; then       // instrucción condiconal con una variable
  echo "Uso: $0 fichero"     // envía mensaje de Uso de fichero
  exit 0                     // sale de condición si cumple
fi                           // termina la instrucción condicional
for i in *.c; do             // estructura de ciclo repetitiva
cat $i >> $1                 // concatena los ficheros
done                         // término de la instrucción repetitiva
```

Este programa concatena todos los ficheros de C en el directorio actual en el fichero que se le pase como primer argumento.

La shell posee muchas sentencias de control de flujo y variables internas que permiten hacer programas muy potentes. Para más información, ver la página de manual de *bash(1)*.

1.6.3 Otros lenguajes

Linux soporta muchos más lenguajes de programación, entre los más populares actualmente están:

- *Java*. Linux es uno de los sistemas recomendados para desarrollar programas en Java, por su gran estabilidad. Hay disponibles varias máquinas virtuales de Java y algunos entornos de desarrollo.

- *Python*. Es un lenguaje interpretado muy fácil y potente. Últimamente está desplazando algo a Perl en la programación de aplicaciones sencillas y rápidas.

1.6.4 Principales conjuntos de herramientas para la creación de interfaces gráficas de usuario.

Librería	Principiante	Licencia	Lenguaje	Vinculados	Ejemplos
TK	Sí	Libre	Tcl	Perl, Python, otros	make xconfig, TKDesk
GTK+	No	Libre (LGPL)	C	Perl, C++, Python, muchos otros	Gnome, Gimp
QT	No	Libre para código abierto	C++	Python, Perl, C,	KDE

Motif	No	Propietaria	C/C++	Pitón	Netscape, Wordperfect

Tabla # 1. *Descripción de herramientas para creación de Interfases Gráficas de Usuario.*

1.6.5 Rendimiento

Hay que tomar en cuenta la velocidad de ejecución de sus aplicaciones cuando se usen comercialmente; las prestaciones dependen más de sus habilidades algorítmicas de programación que del propio lenguaje. Una idea para las pruebas de rendimiento de los lenguajes sería la implementación de un sencillo algoritmo de ordenación en todos ellos y la comparación posterior de los tiempos de ejecución.

1.6.6 Programación Orientada a Objetos (POO)

La Programación Orientada a Objetos es un paradigma de programación importante que ha ganado popularidad. En la programación orientada a objetos, las estructuras de datos y los algoritmos se integran en unidades, a menudo llamadas clases.

La POO contrasta con la programación procedimental (que usa algoritmos y estructuras de datos separados). No depende estrictamente del lenguaje, se puede hacer POO con lenguajes no clasificados como por ejemplo C, y se puede programar en estilo procedimental con lenguajes clasificados como Orientados a Objetos.

1.6.7 Rapid Application Development (RAD, Desarrollo rápido de aplicaciones)

Hay una relación de dependencia de las herramientas que se usan, que del lenguaje propiamente dicho. Para el desarrollo de GUI (Interfaz gráfico de usuario), se basan en la reutilización de código, por lo que el software libre nos puede proporcionar un buen punto de partida.

Como por ejemplo describir los campos de programación en que normalmente se usa el lenguaje en el que se dan otros tipos de usos, buenos y malos, aunque no son tan frecuentes. Y los comentarios, que son información adicional sobre el lenguaje, como son sus capacidades y dialectos.

1.6.8 Lenguajes Principales

Lenguaje	POO	Ejemplos	Comentarios
Perl	Si	Scripts, administración de sistemas, www	Potente para la manipulación de textos y cadenas
Pyton	Si	Scripts, scripts de aplicaciones, www	
Tcl	No	Scripts, administración de sistemas, aplicaciones	
Php	Si	WWW	Popular para bases de datos basadas en web

Java	Si	Aplicaciones para plataformas cruzadas, www	
Lisp	Funcional	Modos de Emacs	Variantes Elisp, Clisp, Écheme
Fortran	No	Aplicaciones Matemáticas	Variantes F77, F90 / 95
C	No	Programación de sistemas, aplicaciones	Muy popular
C++	Si	Aplicaciones	

Tabla # 2. *Principales Lenguajes de Programación en Linux.*

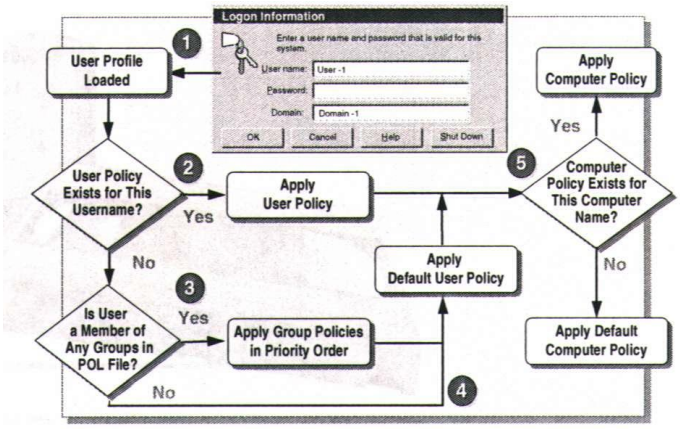
1.6.9 Programación del shell

"Un Shell no es en realidad nada más que un programa diseñado para aceptar comandos y ejecutarlos"[35].

Los shell son también entornos de programación importantes. El conocimiento de los shell es importante para quien trabaje con Linux regularmente, y más aún para los administradores de sistemas.

[35] Linux 4. Edición, Jack Tackett, jr. Steven Burnett, pág. 410

Hay similitudes entre la programación del shell y los scripts, en que a menudo consiguen los mismos propósitos y existe la oportunidad de elegir entre los shell nativos o un lenguaje de scripts. Entre los más populares están los *shell bash, tcsh, csh, ksh y zsh*. Puede obtener información acerca del shell con el comando man; por ejemplo: man bash.



CAPITULO II

ESTUDIO DE INTERFACES DE PROGRAMACION

INTRODUCCION

El manejo de las API's que cada sistema operativo tiene, permite estandarizar las funciones de manejo como cuadros de diálogos, mensajes de error, entre otros. Los niveles de trabajo de cada API establecen una presentación de la aplicación con una manera intuitiva hacia el usuario, agradable y fácil de manejar.

La relación que existe entre Unix – Linux, da un nivel de compatibilidad pero no al cien por ciento como se quisiera, en la mayoría de las aplicaciones se puede establecer un nivel de compatibilidad de acuerdo a las necesidades y configuraciones establecidas por el programador, que ayudan a portar las aplicaciones de un sistema operativo a otro.

2.1 Estudio de la API de Linux

Las normas importantes y especificaciones del API de Linux, incluye una mirada a las normas estandares de la IEEE, y la industria de POSIX. Estos esfuerzos conllevan a la búsqueda de los detalles en el Open Group's Single UNIX.

Los estándares UNIX se centran en la definición del API del sistema utilizando el lenguaje C. El objetivo es que un mismo código fuente pueda ser compilado en dos sistemas POSIX distintos sin necesidad de modificar el código.

Inicialmente se desarrolló una biblioteca que ofrecía el API de PThreads para Linux. Esta implementación se conocía como LinuxThreads. Actualmente, LinuxThreads ha pasado a formar parte de la distribución estándar de la biblioteca de C de GNU, conociada como la "glibc2".

Esta implementación de PThreads se basa en la llamada al sistema clone(2) que Linux ofrece para crear procesos. Utilizando esta biblioteca se pueden desarrollar sistemas de tiempo real blando, ya que las tareas de tiempo real (threads) son ejecutadas sobre un núcleo que no soporta tiempo real estricto. Para la ilustración de esta aplicación se puede observar en la siguiente figura:

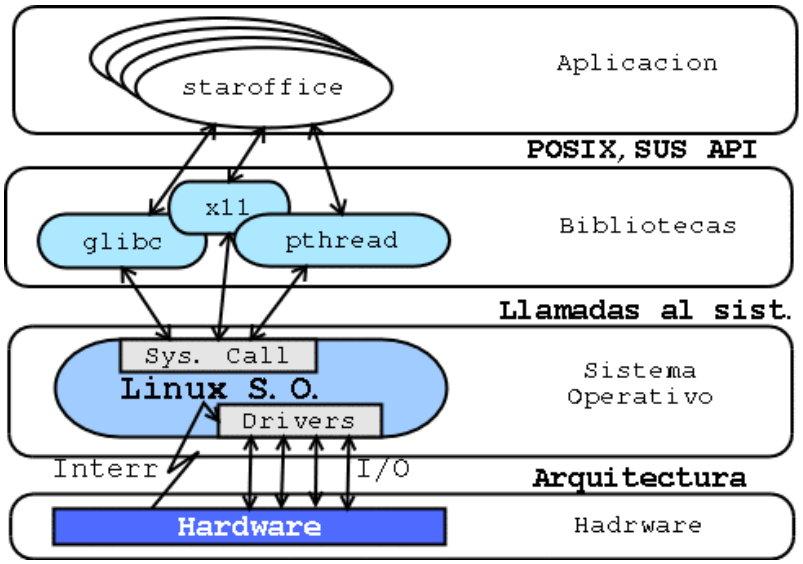


Fig # 2-1. *PThreads en Linux.*

2.1.1 Estandares IEEE – POSIX

El grupo PASC (Portable Application Standards Committee) tiene desarrollado una serie de estándares que forman parte del API de Linux, que son:

- Definiciones Base
- Utilidades y Shell
- Sistemas de Interfases
- Ayuda (Informativo)

Todo este tipo de revisiones fueron probadas para minimizar el número de cambios que conforman cada implementación, de acuerdo a las versiones aprobadas con los

estándares que debían ser requeridos, en consecuencia de resolver conflictos encontrados en versiones anteriores. (Ver apéndice A)

Con estas tareas, Linux se comunica con todo su exterior y entorno es decir, programar sus propios controladores de dispositivos, accediendo directamente a la E/S. Sin embargo, en ocasiones se desea realizar algún tipo de monitorización de lo que sucede al sistema desde Linux, pero comienza con la tarea menos prioritaria. Así, evidentemente, es una manera segura de usar los recursos complejos como el disco (para almacenar datos) o la red.

Linux proporciona para ello las colas de tiempo real (RT-FIFO). Las colas de tiempo real son similares a los pipes nombrados de Unix: existe un dispositivo especial, **/dev/rtfN**, donde pueden leer y escribir con seguridad, tanto tareas de tiempo real como procesos del sistema Linux.

Estos procesos pueden leer y escribir en un RT-FIFO de la manera habitual, abren el dispositivo con la llamada al sistema *open()* y leen o escriben en él con las llamadas respectivas *read() y write().*

2.2 Estudio de la API de Windows y Windows NT

Desde hace pocos años, Microsoft intentaba unificar las dos grandes vertientes de Windows, la serie W9X y la serie NT. Una vez que apareció Windows 3.1, Microsoft

ya tenía definido el API (conjunto de funciones) de programación de Windows prácticamente completo y bastante depurado. Existían dos "pequeños" problemas:

a) todo el subsistema era de 16 bits y además se apoyaba sobre MS-DOS. Realmente no era más que una interfaz (potente) sobre MS-DOS.

b) La multitarea no era real, sino que los programas debían ceder el control al sistema operativo, y una vez que tomaba el control daba paso a la siguiente tarea que tenía en cola.

En ese entonces, el hardware (procesadores 386) ya estaba en el mercado pero totalmente infrautilizado. Este procesador ya soportaba paginación por hardware, switcheo de tareas vía hardware, etc., y Windows 3.1, evidentemente al ser de 16 bits, no utilizaba la potencia que le podía suministrar el procesador.

Por ello, Microsoft se planteó el realizar un verdadero sistema operativo de 32 bits y además utilizando las "features" que le daban los nuevos procesadores de Intel. Microsofot estaba pensando ya en Windows NT.

Con detalle en el kernel de NT, se puede observar que el sistema de archivos nativo ya deja de ser FAT y aparece un nuevo sistema de archivos: el *NTFS*[36]. Esto no solo fue a nivel del sistema de archivos sino también a nivel de la ejecución de tareas siendo ya multitarea real basada en prioridades. Con esto estaba ya constituido el núcleo,

[36] NTFS: Network File System, sistema de archivos para la plataforma NT.

simplemente quedaba implementar el API de 16 bits de Windows 3.1 y convertirlo en 32 bits. Así salió al mercado el primer Windows NT versión 3.1 que rápidamente evolucionó a la versión 3.5 y con unas modificaciones en el núcleo para dar velocidad, de cual evolucionó al Windows NT4.

Cabe recordar, que los procesadores 386 y superiores, ejecutan las tareas en distintos niveles de privilegio. Exactamente en tres niveles de privilegio: 0, 1 y 2 como si fuesen capas de cebolla concéntricas siendo el nivel cero el más inferior.

El nivel 0, es también llamado nivel Kernel, en el que un proceso puede realizar todo acceso al hardware y la ejecución del núcleo del sistema operativo; evidentemente el fallo de un programa en este nivel, tiene por consecuencia la caída inexorable de la máquina.

Los niveles 1 y 2 no se utilizan. Teóricamente, el nivel 1 es donde se ejecutarían los drivers del sistema, pero Microsoft optó por ejecutarlo a nivel Kernel (nivel 0). El último nivel, (nivel 2) es el modo "user", se ejecutan los programas de usuario que realmente no pueden hacer casi nada. Cualquier acceso al hardware o a los recursos del sistema, se debe hacer a través de los niveles anteriores. En este caso a través del nivel 0.

Microsoft empezó a experimentar con la versión de Windows 3.11 (para trabajo en grupo), la conectividad de red y el soporte en redes así como los primeros pasos para

algunos subsistemas de 32 bits dentro del propio Windows (como por ejemplo, acceso a disco en 32 bits y poco más).

Igualmente, creo una capa API de 32 que era capaz de instalarse sobre Windows 3.1 o 3.11 y daba soporte a programas de 32 bits. Este subsistema no era independiente, sino que estaba "montado" por encima del de 16 bits cediendo control a él cuando era necesario. Es decir, era un recubirmiento del API de 16 bits.
Con las experiencias anteriores, se establecen las siguientes mejoras:

- Subsistema de drivers de 32 bits. No están normalizados ya que el sistema basado en VxD (estáticas y dinámicas) posteriormente ha sido abandonado por Microsoft, pero en ese entonces fue una idea realmente buena.
- API de 32 bits totalmente integrada.
- Incluye el TCP/IP como transporte nativo de red por primera vez en la historia de Microsoft.
- Interfaz gráfica mejorada y mucho más amigable que la de versiones anteriores.

A esta evolución, surgió Windows 2000 con la idea de sacar a todos los Windows. Posteriormente se desestimó, haciendo evolucionar únicamente el NT 4 a un sistema más estable y totalmente PnP (Plug and Play).

Windows 2000 se produjo como el sucesor e integrador de todos los Windows. La idea original pasaba por incorporar en Windows 2000 el resto de subsistemas probados y experimentados en la serie de Windows 9X. A lo largo Microsoft replanteó una

transición completa al núcleo NT, y por tanto, el producto final que salió al mercado siguió siendo un NT puro (mejorado en muchas características). La evolución final de W2000 y la integración con algunos de los subsistemas probados con éxito en Windows ME, y la corrección de errores de W2000, nace WindowsXP.

El desarrollo de una nueva "imagen", realiza un cambio en el escritorio, así como sus nuevos efectos visuales. Igualmente, nuevas funcionalidades en donde el usuario domestico se sintiese más a gusto con Windows XP. Entre ellas, una mejora de la capacidad multimedia, capacidad de grabación básica de CD's, cortafuegos personal, soporte de voz (para versiones USA) y otras decenas de funcionalidades que hiciesen a XP un producto apreciable.

2.2.1 API´s General de Windows

La API de Windows es una enorme colección de procedimientos que pueden ser llamados por cualquier programa que corra bajo Windows. Los procedimientos de la API ofrecen: presentar las ventanas en pantalla, usar la impresora, mostrar textos, usar menús..., etc.

Los procedimientos API están localizados en archivos llamados librerías de enlace dinámico, o DLLs. Los archivos DLL se instalan en el disco, y se cargan en memoria cuando las aplicaciones Windows los llama. De igual manera, muchas aplicaciones instalan sus propios archivos DLL para usarlas.

Existe en Visual Basic, una herramienta llamada Visor de Texto API. La documentación completa sobre las API´s (nombres de funciones, tipos y constantes) es inmensamente grande, y buscar cosas en ella sería complicado. Esta herramienta, permite cargar la base de datos de las API´s para buscarlas, y dar la declaración exacta de la función, también permite seleccionar todas las que se quiere para copiar el código.

Una vez cargado el archivo, aparecerá en el campo de elementos disponibles unos 2000 nombres de funciones; cada función es una API diferente. Hay API´s que están relacionadas, que tienen funciones parecidas, y dependen de otras. Estas funciones se distinguen por el prefijo que llevan, tipos de datos propios o constantes.

Por ejemplo, para manejar el MCI (Multimedia) están las funciones mciSendString, mciSendCommand, mciGetErrorString, mciGetDeviceID. Se puede cambiar el campo de tipo de API para desplazarse por los tipos, las constantes y las declaraciones de las API´s.

La colección de funciones API´s han sido realizadas tradicionalmente bajo el dominio de C y C++, de manera que todas las conexiones al API operan de forma más intuitiva para los programadores, dando una gran funcionalidad y mandos en sus sistemas reutilizando su código. (Ver apéndice B).

2.2.2 Funciones del API de Windows.

Como resumen de todas las funciones de la API de Windows, se tiene por ejemplo el cuadro de diálogo que es más utilizado en la mayoría de las aplicaciones, así también como en el mismo Windows.

Como un ejemplo, el API que ayuda a presentar el cuadro de diálogo se encuentra en el directorio **Windows\System** llamado ***Comdlg32.DLL,*** en el que puede presentar cualquier aplicación que llame a esta función API, así como el mismo Windows.

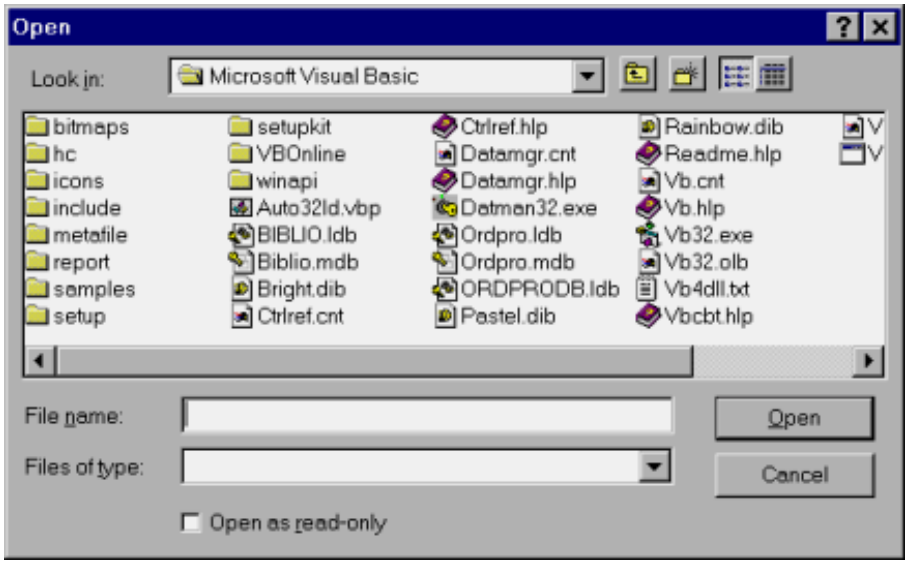


Fig # 2-2. *Cuadro de Diálogo que presenta una función del API de Windows*

Dado el número de llamadas de API que constituyen Windows en general, Microsoft decidió agruparselos en cuatro bibliotecas principales:

- *KERNEL32*

El principal DLL, Kernel32, que ejecuta la manipulación y administración de la memoria, multitasking de los programas que están ejecutando, y la mayoría de las otras funciones que directamente ejecuta Windows.

- *USER32*

Librerías de administración de Windows. Contiene funciones que tratan de los menús, cronómetros, comunicaciones, archivos y muchas otras áreas escondidas de Windows.

- *GDI32*

Interfase de Dispositivo Gráfico. Proporciona las funciones necesarias para dibujos en la pantalla, así como verificar las áreas en las formas a ser nuevamente redibujadas.

- *WINMM*

Proporciona las funciones multimedia, tratando con sonido, música, video en tiempo real, etc. Solamente esta librería es una DLL 32 bits. El equivalente de 16 bits se llama *MMSYSTEM*.

2.3 Estudio de las Librerias Unix como enlace a Linux.

Los orígenes de UNIX se remontan al año 1962 en el que el *CTSS*[37] y el *MIT*[38] investigaron en áreas de tiempo compartido y protección. En 1965, Bell Labs (la división de investigación de AT&T), General Electric y el MIT trabajaron en un macroproyecto llamado *MULTICS* (Multiplexed Information and Computing Service), previsto para desarrollar una gran potencia de cálculo y almacenamiento de muchos usuarios.

De este proyecto, se obtuvieron interesantes resultados (capacidad de multiproceso, árbol de ficheros, shell) pero; como todo proyecto gigante, su complejidad desbordó al equipo que lo emprendió, así que en 1969 fue abandonado.

En el año 1971, Ritchie y Kernigham crean y utilizan el lenguaje C en un PDP-11 (algo así como un AT), lenguaje nacido para la programación de sistemas. Así, dos años después en 1973, Ritchie y Thompson re-escriben "a Unix"en lenguaje C, pero esta vez desarrollan un sistema multiusuario. El sistema, demostró ser algo bueno que ese mismo año Bell Labs contaba con 25 instalaciones funcionando con UNIX.

Entre las universidades a la que llego Unix, fue la Universidad de California en Berkeley. Allí se modificó el sistema incorporando una variante notable la utilización de memoria virtual paginada. Así en 1984 marca un nuevo hito en la historia de Unix

[37] CTSS: Compatible Time Sharing System (Sistema Operativo anterior a Unix)
[38] MIT: Massachussets Intitute of Technology

ya que SUN desarrolla los conceptos de RPC (Remote Procedure Call) y NFS (Network File System) y los incorpora a SunOS.

El objetivo de un Sistema Operativo (S.O.), es crear una máquina virtual para la que sea sencillo trabajar; es decir ocultar el hardware. Por ejemplo, una SUN-10 con SunOS 4.2 (UNIX BSD 4.2 para SUN) y un PC-386 con FreeBSD (UNIX BSD 4.2 para PC) se parecerán iguales al usuario (no en prestaciones). Por el contrario, una misma arquitectura con dos S.O. distintos se presentará como dos máquinas distintas.

Desde una visión más restringida, se considera el S.O. como el núcleo o kernel del sistema con las funciones y estructuras de datos necesarias para gestionar recursos. Así pues, el S.O. debe:

- ejecutar programas (procesos)
- asignar recursos (CPU, memoria)
- encargarse de la protección del sistema
- las operaciones de entrada y salida
- la gestión de usuarios y procesos
- la detección de errores y
- la manipulación del sistema de ficheros.

Tanto Unix como Linux, son sistemas multiusuario y multitarea. Esto influye en la gestión de la protección del sistema que soluciona de la siguiente forma:

- Todas las operaciones de entrada/salida son realizadas en modo supervisor, modo de ejecución en el que el S.O. toma control total del ordenador. El

usuario, sin embargo tiene la impresión de ser él quien realiza la operación invocando una llamada al sistema desde programa.

- Tiene un absoluto control de la memoria, gestionando límites de zona para usuarios y para sí mismo, proporcionando llamadas para petición y liberación.
- Al ser sistemas operativos multiusuarios y multitarea, debe realizar la gestión de la CPU asignando o quitando a los programas de usuario (sistema de tiempo compartido).

La entrada en modo supervisor o activación del S.O. se producirá en tres motivos:

1. Una llamada al sistema
2. Una interrupción (hardware o software)
3. Un trap o interrupción especial hardware que generalmente aniquila el proceso que la provocó

Y como característica importante, permite un grado de particularización del entorno según las preferencias de cada usuario a través de ficheros de configuración particulares.

2.3.1 Ficheros de entrada y salida

Autoconf es una herramienta para producir shell scripts que automáticamente configuran los paquetes de código fuente para adaptarlos a los distintos sistemas. Los scripts de autoconf son independientes de éste, lo cual no necesitan Autoconf para ser ejecutados.

Para cada paquete de software Autoconf crea un script de configuración partiendo de un patrón con una lista de características del sistema que usa o que puede usar. La intención que se tiene es generar un software portable. Autoconf requiere GNU m4

2.3.2 Creando configure Scripts

El fichero que genera autoconf se llama por convención *"***configure***"*:

- Uno o más makefiles, uno en cada subdirectorio del paquete
- Opcionalmente un fichero de cabeceras conteniendo directivas *#define*
- Un script de shell denominado config.status, que cuando se ejecuta puede volver a crear los ficheros
- Un script llamado config.cahe, que guarda la información de muchos de los test que se han ido ejecutando
- Un fichero llamado **config.log**, que guarda todos los mensajes producidos por los compiladores.
- Para crear el fichero **configure**, es necesario escribir el fichero de entrada configure.in y sobre él ejecutar autoconf, también se puede escribir ficheros como: *aclocal.m4, acsite.m4* para inclur macros propias acconfig.h para incluir sus propios ficheros de cabeceras. El orden en que autoconf llama a las distintas macros son:

```
AC_INIT(file)                    //archivo ac… de entrada (aclocal.m4, acsite.m4)
checks for programs              // inspección de programas
```

checks for libraries	// inspección de librerías
checks for header files	//inspección de archivos de cabecera
checks for typedefs	//inspección de tipos de definiciones
checks for structures	//inspección para estructuras
checks for compiler characteristics	//inspección de características del compilador
checks for library functions	//inspección de funciones de librerías
checks for system services	//inspección de servicios del sistema
AC_OUTPUT([file...])	// archivo ac… de salida o generado

En el caso supuesto, en que el archivo **"configure.in"** no exista, hay la utilidad o la macro llamada autoscan en el que puede crear un **configure.in**, que busca en los ficheros fuentes problemas de portabilidad común y crea el fichero configure.scan.

2.3.3 Inicialización y fichero de entrada

Cada fichero **configure.ini** debe llamar a *AC_INIT* antes de empezar a compilar. Su sintaxis es la siguiente:

AC_INIT (fichero_único_en_directorio_fuente)

AC_INIT contiene un solo fichero y esta macro comprobará su existencia y cuando la compruebe toma ese directorio como directorio *'top-level'* y a partir de él encuentra todos los demás.

2.3.4 Creando ficheros de salida

Cada fichero autoconf generado debe acabar llamando a *AC_OUTPUT*. Esta macro lo que hace es generar los ficheros makefiles y otros opcionales resultantes de la configuración. Su sintaxis es la siguiente:

AC_OUTPUT ([file... [, extra-cmds [, init-cmds]]])

Esta macro crea los ficheros de salida separados por espacios en blanco. Esta macro crea los ficheros citados a partir de unos '.in' del mismo nombre. Los ficheros típicos a generar son los makefiles entre otros.

Ejemplo: AC_OUTPUT(Makefile src/Makefile man/Makefile X/Imakefile)

2.3.5 Sustituyendo valores de variables de salida.

Un paquete de software debería distribuir también su configure.in para que en caso necesario un usuario pueda hacer las modificaciones pertinentes para adaptarlo a su máquina. Sin embargo, lo que no es necesario es distribuir los Makefiles porque estos se generan en el **./configure**.

2.3.6 Sustituciones en Makefiles

Cada directorio tiene un Makefile que existe para su revisión, compilación, e instalación; estos Makefiles nacen de los **Makefiles.in**. Después de que el makefile generado es ejecutado mediante el comando make, va recursivamente actuando sobre los distintos makefiles de los subdirectorios.

Las variables que aparecen dentro del archivo Makefiles.in son de tipo *@variable@* el cual son sustituidas por sus valores en los makefiles finales, en el que estas variables son entendidas por el shell. La sustitución de las variables se hace con la macro *AC_SUBST* cuando se ejecuta **./configure**. Los valores definitivos en los makefiles se denominan variables de salida.

2.3.7 Variables de salida

En el cuadro que se presenta a continuación, se encuentran algunas variables de salida que describe su funcionamiento y en el que reconoce la macro autoconf.

Variable	Descripción
Bindir	Directorio para instalar aplicaciones que los usuarios ejecutan
Datadir	Directorio en el que se instalan datos independientes de la arquitectura y es solo de lectura. Un ejemplo pueden ser ficheros .gif
Libexecdir	Directorio para instalar ejecutables, que otros programas ejecutan
Localstatedir	Directorio para instalar datos modificables de una simple máquina

Mandir	Directorio de nivel más alto para instalar documentación en formato man
Prefix	Directorio de instalación principal
Sbindir	Directorio donde se instalan los ejecutables que el administrador del sistema ejecuta
Srcdir	Directorio donde se ubicarán los fuentes
Sysconfdir	El directorio para instalar datos de solo lectura de una máquina simple

Tabla #3. *Descripción de Variables de Salida.*

2.3.8 Construyendo directorios

Se puede compilar un mismo paquete para varias arquitecturas simultáneamente. Esto puede hacer mediante la variable *VPATH*. Si se desea que cada paquete utilize ficheros **configure** diferentes, sus **configure.in** también lo serán y hay que definir en los Makefile.in estas dos líneas.

srcdir = @srcdir@

VPATH = @srcdir@

Configure se encargará de sustituir el valor correcto para scrdir cuando cree el Makefile.

2.3.9 Archivos de entrada y salida

Se denomina al archivo de entrada Makefile.am y al fichero de salida Makefile.in Automake es una macro que permite automatizar la creación de ficheros make (los makefiles). El mayor atributo de Automake es que reduce considerablemente el trabajo de mantenimiento de estos makefiles.

Automake convierte los ficheros Makefile.am a Makefile.in. En el que makefile.am son una serie de instrucciones y reglas que son requeridas en unos casos y en otros no. Debe existir un makefile.am en cada directorio o subdirectorio del software a distribuir y la información que contiene es relativa a ese directorio que representa.

Por lo tanto, quiere decir que habrá tantos Makefile.am como directorios haya en la distribución del software. Además Automake utiliza un repositorio parecido al de cvs que no es estándar para la gestión y control de los ficheros make.

Los ficheros *".am"* contienen comentarios, etiquetas y variables. Los comentarios empiezan con "#" y las variables como en el shell. Por ejemplo subdirs es una variable que contiene los directorios que hay dentro del directorio actual:

Ejemplo: SUBDIRS src lib po

Esto da información para el posterior análisis de la estructura de directorios.

2.3.10 Tipos estructuras de directorios

Automake reconoce 3 tipos de estructuras de directorios

- *Flan*: Todos los ficheros que componen la distribución se concentran en un directorio que no contiene subdirectorios.
- *Shallow*: En el directorio principal se encuentra los fuentes del programa principal, mientras que el resto se ubican en un determinado subdirectorio
- *Deep*: En el directorio principal se tienen la configuración del paquete que se va a distribuir; por ejemplo los fuentes pueden estar ubicados en un subdirectorio concreto.

2.3.11 Opciones más importantes de Automake

--*include-deps*

Esta instrucción incluye automáticamente toda la información de dependencias generadas.

--*generate-deps*

Genera un fichero concatenando automáticamente toda la información de las dependencias generadas en un fichero `.dep_segment'.

--*output-dir=dir*

Ubica el archivo **Makefile.in** en el directorio especificado por 'dir'. Si no se establece la ruta o el path se crea en el mismo directorio donde está en **Makefile.am**.

2.3.12 Esquema general

En el gráfico siguiente se resume el enlace de Unix a Linux para el funcionamiento de alguna aplicación.

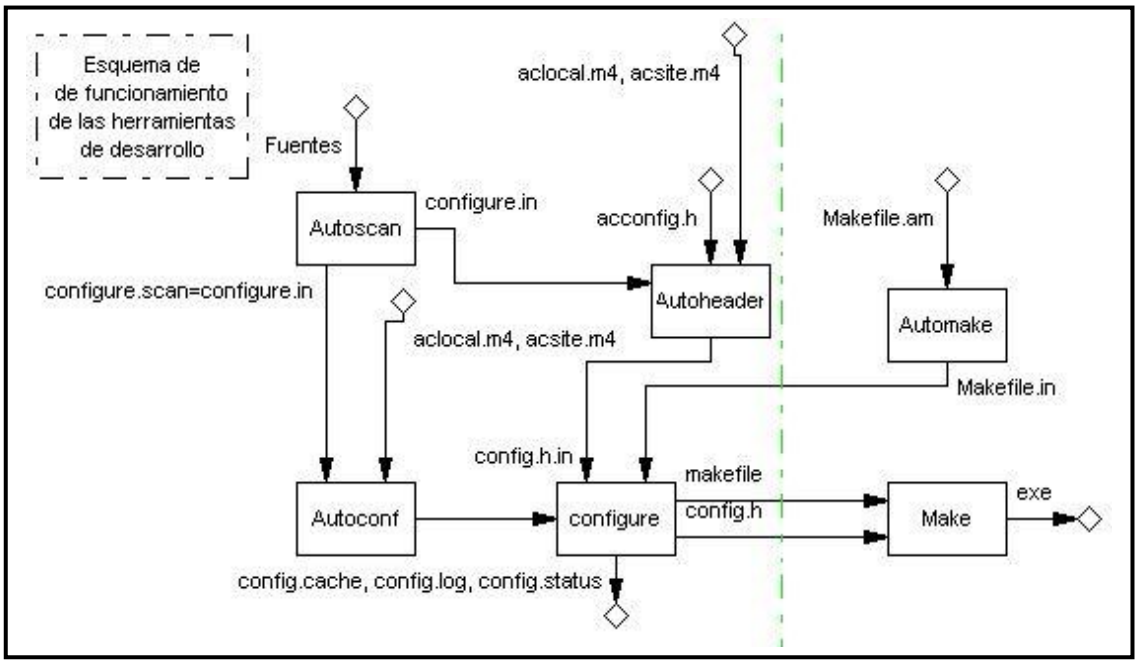


Fig # 2-3. *Esquema del enlace Unix a Linux.*

2.3 Análisis en la Implementación de Widgets

Los widgets son tipos de componentes para poder crear interfases gráficos. Los widgets pueden comportarse de una determinada forma dependiendo de los eventos que proporciona el usuario. Hay widgets que son ventanas, botones, cajas de texto, etc.

La estructura básica que forma la mayoría de todos los widgets son:

- Constructor y los Métodos de Componer: estos son utilizados para acumular toda el dato relevante para exponer un conjunto de capas de alguna manera.
- Método Build: toma toda la información y pone junta a los CSS[39] y DIV separadamente en dos propiedades: **.css** y **.div**
- Método Activate: asigna todas las Dynlayers y los eventos necesarios para que el widget funcione.
- Escribe las CSS y DIV's: utilizando las propiedades CSS y DIV con la propiedad document.write() y luego a la página.

2.4.1 Otros Tipos de Widgets

Los widgets a estudiarse hacen referencia a la librería de GTK, para Linux, ya que tienen una similitud general para todas las librerías a estudiarse.

[39] CSS: Cascade Style Sheet

2.4.1.1 El widget árbol.

El propósito del widget GtkTree es mostrar datos organizados de forma jerárquica y trabaja en su propia ventana. GtkTree en sí mismo no es muy diferente de GtkList, ambos están derivados directamente y funcionan igual. La diferencia es que los widgets GtkTree pueden anidarse dentro de otros widgets GtkTree.

2.4.1.2 El widget Botón

2.4.1.2.1 Boton Normal

Es aquel que se crea con una etiqueta en forma de rectángulo, estableciendo atributos de pulsado, color, etiqueta, etc.

2.4.1.2.2 Botones de selección (Toggle Buttons)

Estos botones son muy similares a los normales. La única diferencia es que sólo pueden estar en dos posiciones diferentes alternadas mediante pulsaciones del ratón. Los botones de selección son la base de otros tipos: los de comprobación y los circulares, por lo tanto muchas de sus llamadas serán heredadas por estos.

2.4.1.2.3 Botones de comprobación

Los botones de comprobación son un poco diferentes a los anteriores, aunque sus propiedades y funciones son bastante similares. En lugar de ser botones con texto en su interior son pequeños cuadrados con texto a su derecha. Normalmente son usados para (des)seleccionar opciones.

2.4.1.2.4 Botones circulares

Estos botones son similares a los de selección con la salvedad de que están agrupados, de modo que sólo uno puede estar seleccionado. Por tanto son usados para permitir al usuario seleccionar algo de una lista de opciones mutuamente excluyentes.

2.4.1.3 El widget EventBox

Algunos widget gtk no tienen asociada una ventana X, por lo que sólo pueden dibujar en la ventana padre. Debido a esto, no pueden recibir ningún evento y si tienen un tamaño incorrecto, no se recortarán correctamente por lo que se sobrescriben en ciertas zonas.

2.4.1.4 El widget GtkCList

El widget GtkCList ha reemplazado al widget GtkList (que sigue estando disponible).
El widget GtkCList es un widget de una lista multicolumna que es capaz de manejar, literalmente, miles de filas de información. Cada columna puede tener (opcionalmente) un título, que puede estar activado (opcionalmente), permitiéndonos enlazar una función con la selección.

2.4.1.5 El widget lista

El widget GtkList está diseñado para actuar como un contenedor vertical de widgets que deben ser del tipo GtkListItem.

Un widget GtkList tiene su propia ventana para recibir eventos y su propio color de fondo, que normalmente es blanco.

2.4.1.6 El widget GtkListItem

El widget GtkListItem está diseñado para comportarse como un contenedor que tiene un proceso hijo, proporcionando funciones para la selección / deselección justo como las necesitan los procesos hijos del widget GtkList.

2.4.1.7 El widget menú

Esta clase de widget construye los menús para las aplicaciones o accesos a programas de manera interctiva. Se puede encontrar de dos maneras: 1) el menú normal o de barras y 2) el menú contextual o flotante.

2.4.1.8 El widget texto

El widget texto permite mostrar y editar múltiples líneas de texto. Admite texto en varios colores y con varios tipos de letra, permitiendo mezclarlos de cualquier forma que desee. También hay un gran número de teclas para la edición de textos, que son compatibles con Emacs. El widget texto admite cortar, copiar y pegar, dentro de las funciones de cada sistema operativo.

2.4.1.9 Widgets Contenedores

2.4.1.9.1 Libros de notas (Notebooks)

El widget Notebook es una colección de “páginas” que se ocultan las unas a las otras, cada una con un contenido diferente. Este widget se ha vuelto cada vez más común últimamente en la programación de interfaces gráficos de usuario (GUI), y es una forma de mostrar bloques de información similar que necesitan aparecer de forma separada.

2.4.1.10 Ventanas con barras de desplazamiento

Las ventanas con barras de desplazamiento se utilizan para crear una zona de movimiento dentro de una ventana real. Puede insertar cualquier tipo de widget en una ventana con barras de desplazamiento.

2.4.1.11 El widget "ventana dividida" (Paned Window)

El widget ventana dividida es útil para cuando se quiere dividir una zona en dos partes, con un tamaño relativo controlado por el usuario. Entre las dos porciones de la ventana se dibuja un separador con un botoncito que el usuario puede arrastrar para cambiar el tamaño de cada zona. La división puede ser horizontal (HPaned) o vertical (VPaned).

2.4.1.12 Barras de herramientas

Las barras de herramientas acostumbran a agrupar un conjunto de widgets para hacer más sencilla la personalización de su aspecto y composición. Típicamente una barra de herramientas consiste en botones con iconos, etiquetas y tips para los iconos (pequeño texto descriptivo que aparece cuando se mantiene el ratón sobre el icono), pero en realidad en una barra se puede poner cualquier tipo de widget. Finalmente, los elementos se pueden disponer de forma horizontal o vertical, y los botones pueden mostrar iconos, etiquetas o ambos.

2.4.1.13 Marcos con proporciones fijas

El widget aspect frame (marco proporcional) es como el widget frame (marco), excepto que conserva las proporciones (la relación entre el ancho y el alto) del widget hijo, añadiendo espacio extra en caso de ser necesario. Esto es útil, por ejemplo si se quiere

hacer una vista previa de una gran imagen. El tamaño de la vista previa debería variar cuando el usuario redimensione la ventana, pero la proporción tiene que coincidir con la de la imagen original.

2.4.1.14 Widgets de selección de rango

Este tipo de widgets incluye a las scroollbar (barras de deslizamiento) y la menos conocida scale (escala). Ambos pueden ser usados para muchas funciones y su implementación son muy parecidas. Principalmente se utilizan para permitirle al usuario escoger un valor dentro de un rango ya prefijado.

Todos los widgets de selección comparten elementos gráficos, cada uno de los cuales tiene su propia ventana XWindow y recibe eventos. Todos contienen una guía y un rectángulo para determinar la posición dentro de la guía (en un procesador de textos con entorno gráfico se encuentra situado a la derecha del texto y sirve para situarnos en las diferentes partes del texto). Con el ratón podemos subir o bajar el rectángulo, mientras que si se hace `click' dentro de la guía, pero no sobre el rectángulo, este se mueve hacia donde se ha hecho el click. Dependiendo del botón pulsado el rectángulo se moverá hasta la posición del click o una cantidad prefijada de ante mano.

2.4.1.15 Widgets de escala

Los widgets de escala se usan para determinar el valor de una cantidad que se puede interpretar visualmente. El usuario probablemente fijará el valor al ojo. Por ejemplo el widget color contiene escalas que controlan los componentes del color a seleccionar.

Normalmente el valor preciso es menos importante que el efecto visual, por lo que el color se selecciona con el ratón y no mediante un número concreto.

2.4.1.16 Widgets de rango vertical

Todos los widgets de rango pueden ser manipulados con las teclas arriba, abajo, Re Pág, Av Pág. Las flechas mueven las barras la cantidad fijada mediante los incrementos que estén establecidos.

2.4.1.17 Widgets de rango horizontal

Las teclas izquierda y derecha funcionan tal y como desea el usuario, mueven la barra una cantidad dada por pasos de incremento. A su vez inicio y final sirven para pasar de un extremo al otro de la guía. Para el widget GtkHScale el mover la barra una cantidad dada por incremento de página se consigue mediante Control-Izquierda y Control-derecha, mientras que para el widget GtkHScrollbar se consigue con Control-Inicio y Control-Final.

2.4.1.18 El widget de información rápida (tooltip)

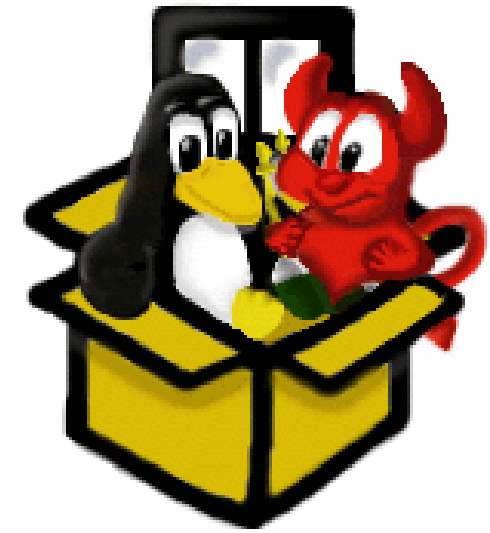
Estos widgets son las pequeñas etiquetas que texto que aparecen cuando se sitúa el puntero del ratón sobre un botón u otro widget durante algunos segundos. Es bastante fácil de usar.

2.5 Conclusión

Las API´s de los sistemas operativos son un conjunto de rutinas que utiliza una aplicación para solicitar y realizar servicios de bajo nivel efectuados por un sistema operativo. Estas rutinas generalmente llevan a cabo tareas de mantenimiento como administrar archivos y mostrar información.

Para establecer un enlace entre Linux – Unix los comandos necesarios son: autoconf y automake los cuales se encargan de la elaboración y generación de archivos, en los que se adaptan a la configuración del nuevo sistema, para la compilación de las aplicaciones.

El manejo de todos los entornos de los sistemas operativos se realizan a través de widgets, en el que el usuario interactúa de una manera fácil y cómoda.



CAPITULO III

ESTUDIO Y ANALISIS DE EMULADORES

INTRODUCCION

Una de las grandes barreras para el uso de Linux es la dificultad de seguir aprovechando las aplicaciones de Windows. Los entornos avanzados y profesionales llevan a un estudio de *"compatibilidad"*; abarcando carácterísticas propias de una plataforma o de un sistema operativo en el que se puede hablar de emulación.

Los emuladores habituales o más conocidos son Wine y DosEmu. Wine a pesar de no ser calificado como un emulador, es capaz de ejecutar algunas aplicaciones Windows dentro de Linux, traduciendo las llamadas API's de Windows en llamadas propias de Linux. En cambio con Dosemu, es capaz de emular un gran número de aplicaciones DOS dentro de Linux, dejando a un lado los sistemas gráficos.

3.1 Estudio y análisis de los distintos emuladores como Wine y DosEmu.

3.1.1 WINE

Wine es un proyecto de fuentes abiertas es decir, los archivos fuente están disponibles para cualquier programador que desee contribuir y/o modificar estas implementaciones. Aunque es un proyecto enfocado principalmente hacia la plataforma Linux, existen adecuaciones para que se ejecute en plataformas variantes de Unix como: FreeBSD, OpenBSD, NetBSD, Solaris, además de algunos trabajos hechos en SCO Unix OpenServer, SCO Unixware y el OS/2 de IBM.

El proyecto Wine se dirige en dos direcciones:

a) La primera consiste en desarrollar una serie de librerías que funcionen igual que las API de Microsoft, de manera que los programas compilados (binarios) de Windows envíen peticiones y reciban respuestas, de la misma manera que las esperadas dentro de una plataforma Windows; con esto, una gran cantidad de aplicaciones de Windows pueden ejecutarse sin dificultad en un ambiente XWindows de Linux, y el conjunto de soluciones se incrementaría de manera automática.

b) La segunda dirección del proyecto Wine se encamina a desarrollar API nativas en Linux, de modo que los programadores puedan compilar sus programas con estas librerías y obtener aplicaciones nativas de Linux, con un mínimo de modificaciones en sus programas, y un esfuerzo casi nulo.

Además, las aplicaciones así compiladas pueden utilizar todas las ventajas y los recursos que ofrece Linux. Debido a esto, la disponibilidad de aplicaciones nativas para

Linux, enfocadas a una gran diversidad de áreas de interés, se incrementaría con rapidez.

Wine no es un emulador de Windows, ya que no pretende simular una máquina, sistema operativo y librerías (como sería el caso de emular Windows sobre una computadora PowerPC con sistema operativo de Macintosh). De hecho, Wine utiliza el mismo hardware que Windows, y en realidad sólo sustituye las librerías a las que hacen referencia los programas, por lo que el desempeño y la funcionalidad casi son los mismos que se obtendrían sobre Windows.

Tampoco es la panacea. Wine nunca ejecutará todos los programas desarrollados para Windows y Dos, pues los programas que no se escribieron con apego a las especificaciones, o utilizan llamadas a funciones no documentadas, tendrán problemas para ejecutarse, aunque porten las llamadas a librerías API estándar y algunas otras funcionalidades. Sin embargo, la mayor parte de las aplicaciones recientes y bien desarrolladas funcionarán sin problemas en Linux con Wine.

3.1.1.1 La Instalación de WINE

Wine se diseña para correr sobre cualquier Computadora x86. Un mínimo de 32 MB la memoria real (se recomienda 64MB) y un mínimo espacio Swap de 64 MB. Wine requiere de 20 MB de espacio mínimo en disco para la versión estándar. Se necesita Linux instalado, FreeBSD o Solaris sobre su sistema con sistema de X Windows; no necesita Windows para correr sus aplicaciones en wine.

Para descargar el programa Wine se recurre a la Web oficial www.winehq.com. Wine es una serie de librerías capaces de ejecutar programas Windows (Windows 3.x , Windows 9x, Windows NT, Windows 2000 y actualmente Windows XP); con las API´S de Unix/Linux.

Tiene soportes de llamadas de Win16 y Win32; para código 16 y 32 Bits, de igual manera para drivers de impresoras Win 3.1, juegos basados en DirectX, puertos seriales, entre otros. Para ejecutarlo wine se debe estar como usuario root, o si es usuario alternativo, debe tener los permisos necesarios para la ejecución.

Una vez extraído todos los archivos y colocado en sus respectivos directorios, se ejecutan los scripts de instalación y se procede a la modificación del archivo de configuración wine.conf con cualquier editor.

Este archivo se puede encontrar en: **/etc/wine , /etc/ o /usr/local/etc**, o en las nuevas versiones: **/root/.wine o /home/.wine**.

3.1.1.2 Configuración de Wine.

Se edita el archivo wine.conf, en el cual aparecen las instrucciones que nos ayudan a ejecutar algunas aplicaciones de windows.

```
[Drive A]            // Describe el drive A
Path=/mnt/floppy     // Ruta donde se encuentra el floppy
```

Type=floppy // Tipo de dispositivo

Label=Floppy // Etiqueta del dispositivo

Serial=87654321 // Se establece como el serial del dispositivo

Device=/dev/fd0 // Dispositivo como lo carga el sistema operativo

[Drive C] // Describe al Disco Duro

Path=/Wine/ // Establece el camino donde esta Wine

Type=hd // El tipo de dispositivo, disco duro

Label=MS-DOS // Etiqueta

Filesystem=win95 // Tipo de archivo de sistema que soporta

[Drive C]

Path=/mnt/discoc/windows

Type=hd

Label=(Etiqueta de windows)

Filesystem=win95

:

[Drive D] // Describe al CDRom

Path=/mnt/cdrom // Camino donde se encuentra montado el CDRom

Type=hd // El tipo de dispositivo

Label=cdrom // La etiqueta del dispositivo

Serial=87654321 // Su numero serial

Device=/dev/hdb // Dispositivo como lo reconoce el sistema operativo

Apartado: [wine] // Ejecución de wine

Path=c:\windows;c:\windows\system;d:\;d:\test;e:\ // Establece el camino donde se encuentran las librerías de windows para su enlace en caso de que no encontrar alguna Api o librería de windows.

A continuación se encuentran los apartados # <wineconf>, [DllDefaults], [DllOverrides] y [x11drv] que se recomienda dejar como están ya que su modificación requieren práctica en el uso de wine.

3.1.1.3 Tipos de DLL que maneja Wine.

En Wine existe referencia de enlaces a las librerías dinámicas de windows, que se describe a continuación:

- *Native*: Es un DLL que fue escrita por Microsoft Windows
- *Builtin*: Es un archivo o librería de WINE, que pueden formar parte de libwine.so, o la librería más actual; es un archivo especial .so que WINE puede habilitar o cargar al momento de una petición.
- *So*: Es un archivo nativo de Unix (.so), en el que genera la llamada a conversión cuando la librería es cargada o llamada. Esto es principalmente útil para las librerías tal como "glide" que tiene exactamente la misma API en ambos Windows y Unix
- Las [DllDefaults], especifica a Wine en que orden debe buscar los tipos de Dll disponibles .

- Con [DllOverrides], se especifica a las Dll´s llamadas, para ser manipuladas, en particular si se desea utilizar Dll´s "native" o no, o si se tiene la misma configuración real de Windows.

Al llegar al apartado [fonts] en el que podemos establecer las fuentes de letra que wine usará; así como su resolución. Ejemplo:

Resolution = 96

Default = -adobe-times-

En los siguientes apartados [serialports] y [paralleports] se especifica los dispositivos conectados a nuestro PC como modems e impresoras. En [serialports] busca en nuestro sistema el puerto del modem y la velocidad en el que se especifica del siguiente modo:

[serialports] // Establece los puertos seriales

Com1=/dev/ttys0

Com2=/dev/ttys1

Com3=/dev/ttyS2

Com4=/dev/modem,38400

[parallelports] // Establece el Puerto paralelo

Lpt1=/dev/lp0

Tweak.Layout]

WineLook=win98 (también podemos poner en su lugar win31 o win95) // Se refiere al tipo de sistema operativo.

Terminada la configuración del archivo wine.conf se guarda, para su ejecución.

Para ejecutar Wine en terminal:

 # cd /

 # cd /discoc/windows (entrar al directorio donde esta windows)

 # wine freecell.exe

Si ejecuta el juego Carta Blanca, se establece una buena configuración.

3.1.2 Dosemu

"Dosemu es un programa a nivel de usuario que utiliza algunas características especiales del kernel de Linux y del procesador 80386 para correr MS-DOS en lo que podría llamarse una ventana del DOS o Dos *box*".

Dicha ventana, tiene las siguientes capacidades:

- La habilidad de virtualizar todas las entradas/salidas y las instrucciones de control del procesador.
- La habilidad de soportar el tamaño de palabra y los modos de direccionamiento del "modo real" de la familia de procesadores x86 mientras que se ejecute entre los límites del modo protegido.
- La habilidad de canalizar todas las llamadas al sistema del DOS y la BIOS, emulando dichas llamadas que funcionen de forma apropiada y se obtenga un buen rendimiento.
- La habilidad de simular un esquema *hardware* sobre el que las aplicaciones del DOS están acostumbradas a tener control.

- La habilidad de proporcionar servicios del MS-DOS a través de las prestaciones nativas del Linux; por ejemplo, dosemu puede proporcionar un disco duro virtual perteneciendo éste a la jerarquía de directorios de Linux.

3.1.2.1 Principales problemas durante la compilación e instalación de dosemu.

A continuación se detalla algunos problemas en dosemu:

1. Olvidar leer el fichero QuickStart.
2. Intentar compilar con un kernel de versión anterior.
3. Tener mal la ubicación del código fuente del kernel de linux, debe estar en **/usr/src/linux.**
4. Compilar con un gcc o una libc en versiones anteriores
5. Olvidar editar el fichero **/etc/dosemu.conf.**
6. Correr DOSEMU con acceso a las particiones cuando éstas ya estén montadas.
7. No instalar dosemu con privilegios suficientes (p.e., root).
8. Intentar correr DOSEMU en un ambiente multiusuario.

3.1.2.2 Compilar dosemu.

Para compilar Dosemu, puede hacerlo como usuario o como root, aunque para usuario hay que dar los permisos respectivos de ejecución.

A la hora de acceder a los puertos de entrada/salida (incluyendo a la consola) dosemu necesita ser ejecutado como root. Ejecutar dosemu en un xterm o en X Window y necesitando acceso directo al hardware le permite ejecutarlo como usuario.

3.1.2.3 Utilización del disco duro con dosemu

Primeramente, monta las particiones del Dos como subdirectorios de Linux. Por ejemplo: crear un directorio en Linux tal como /dos

mkdir -m 755 /dos

y añadir una línea como la siguiente:

/dev/hda1 /dos msdos umask=022

en **/etc/fstab**. (En este ejemplo, el disco duro está montado sólo lectura. Puede montarlo como lectura/escritura reemplazando "002" por "000" y utilizando la opción -m 777 con mkdir). Realizar un mount /dos, de forma que pueda añadir una línea como la siguiente:

lredir d: linux\fs/dos

en el archivo AUTOEXEC.BAT en su hdimage. En un sistema multiusuario, puede utilizar:

lredir d: linux\fs\${home}

donde "home" es el nombre de una variable de entorno que contiene la localización del directorio del Dos (/dos en nuestro ejemplo).

3.1.2.4 Dosemu en un sistema multiusuario

Si se utiliza dosemu en un sistema multiusuario en el que más de un usuario puede ejecutar dosemu, se deberá cambiar el directorio de su imagen de disco duro.

En el archivo **/etc/dosemu.conf** existe una línea de forma predeterminada que indica que la imagen del disco duro es "hdimage". Si lo cambia por **/var/lib/dosemu/hdimage** entonces los usuarios no tendrán que preocuparse por el directorio en el que estén cuando ejecuten dosemu, y el archivo hdimage no necesitará ser cambiado cada vez que instale una versión nueva de dosemu.

En el caso de que esto ocurra para un Dosemu multiusuario, entonces se crea el archivo hdimage de sólo lectura en **/var/lib/dosemu** para cualquier usuario menos para el administrador del Dosemu.

Hay que tener en cuenta que se puede usar el nuevo controlador emufs.sys para montar un directorio "público" y/o un directorio "privado" (un subdirectorio en el directorio home de cada usuario).

Los usuarios deben también crear un fichero de configuración personal llamado ~/.dosrc (con el mismo formato que el /etc/dosemu.conf) para ejecutar su copia personal del dos.

3.2 Estudio de Wise (Windows Interface Source Environment) para Win NT

Debido a que, para el estudio de este capítulo se presentó el la revista PC Actual de enero de 1999 se habla sobre Wise, pero no existe ninguna información acerca de este emulador, ya que no es un emulador , sino una interfaz.

3.3 Estudio de diversos lenguajes de prototipado.

3.3.1 Definición de prototipo.

Un prototipo es un modelo (representación, demostración o simulación) fácilmente ampliable y modificable de un sistema planificado, probablemente incluyendo su interfaz y su funcionalidad de entradas y salidas.

3.3.2 Lenguajes de Prototipado Exploratorio, Experimental y Operacional

- *Exploratorio*: este tipo de lenguje es utilizado para clarificar las metas del proyecto, identificar requerimientos, examinar alternativas de diseño o investigar un sistema extenso y complejo.
- *Experimental*: utilizada para la validación de especificaciones de sistema
- *Operacional*: lenguaje iterativo que es progresivamente refinado hasta que se convierte en el sistema final de desarrollo.

3.3.3 Tipos de Herramientas de Lenguajes de Prototipado

De menor a mayor formalidad:

- Papel y lápiz
- Software de dibujo (widgets)
- Aplicaciones para desarrollar demos
- Software de animación y presentaciones (ej: gifs animados)
- Perl + Motif + Tcl/Tk (UNIX)
- Herramientas visuales para RAD, como Visual Basic, Optima ++ y Borland Delphi
- 4GLs
- Sistemas de gestión de interfaz de usuario (UIM)
- Lenguajes de especificación ejecutable (C, C++)

3.3.4 Éxito o Fracaso de un Lenguaje Prototipado

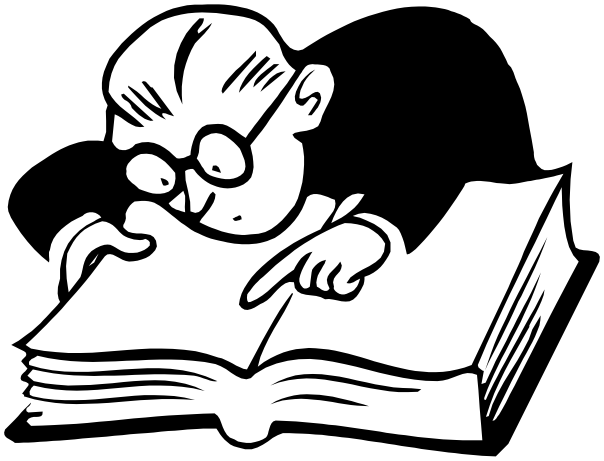
El lenguaje de prototipado tendrá éxito (probable) cuando se utilice

- para comparar alternativas de diseño
- para crear una especificación activa en constante evolución
- para evaluar las interfaces propuestas
- para explorar los efectos de las peticiones de cambio
- para mostrar nuevos o inesperados requerimientos

- para modelar un sistema con una componente significativa de interfaz de usuario
- para modelar un sistema relativamente grande y complejo
- dentro de una herramienta de lenguaje de prototipado rápido

El lenguaje de prototipado tendrá fracaso (probable) cuando:

- no se haya establecido un claro criterio de conclusión del ciclo iterativo de desarrollo
- cuando la funcionalidad de un lenguaje de prototipo operacional no evolucione hacia la aplicación del sistema final
- las opiniones de desarrolladores divergen durante la fase de iteración
- el proyecto es demasiado pequeño para justificar el lenguaje de prototipado
- el lenguaje de prototipado es inapropiado para el producto objetivo
- se utilice para modelar sistemas que no presentan interfaz externa



CAPITULO IV

METODOLOGIA DE APLICACIONES

INTRODUCCION

La alternativa para hacer programas multiplataforma, especialmente en lo que respecta a la interfaz gráfica es utilizar lenguajes interpretados como Java, J++ en diferentes plataformas, con el que se puedan enlazar los componentes. De igual manera la utilización de C, C++ que se puede utilizar con mayor eficiencia donde se pueden incluir librerías e infinidad de módulos disponibles.

Los análisis que se realizan al verificar el uso de librerías estáticas o dinámicas dan una forma de trabajo adecuada a las aplicaciones, a la vez una tranparencia directa para el usuario.

4.1 Recompilación de código fuente

4.1.1 Recompilación estática

En esta técnica se toma un programa escrito en el código emulado y se intenta traducir éste al código ensamblador de la computadora, el resultado será un ejecutable que puede hacer funcionar en la computadora sin ninguna herramienta especial. Es posible combinar la recompilación estática con un intérprete o recompilador dinámico.

4.1.2 Recompilación dinámica

La recompilación dinámica es esencialmente la misma que la estática, pero ocurre durante la ejecución del programa. En lugar de intentar recompilar todo el código de una vez, se puede saltar hacia una instrucción CALL o JUMP. Para incrementar la velocidad, esta técnica puede ser combinada con la recompilación estática.

4.2 Estudio de decompiladores para Windows

La decompilación es lo inverso de la compilación, traduciendo un archivo ejecutable en un lenguaje de nivel más alto. La decompilación es muy útil si las fuentes originales no están disponibles.

En cuanto a los decompiladores, ofrecen la posibilidad de reconocer el tipo de compilador con el que ha sido creado el ejecutable y reconocer a su vez funciones,

variables y eventos de programa propios del lenguaje en que ha sido escrito; que de otra manera sería posible ver con desensambladores. Son extremadamente útiles si se sabe utilizar.

4.2.1 Historia.

Los decompiladores han sido descritos por una variedad de aplicaciones desde el desarrollo de los primeros compiladores. El primer decompilador fue escrito por Joel Donnelly en 1960 en el Naval Electronic Labs, para decompilar en código máquina a Neliac en una computadora Remington Rand Univac M-460 Countess. El proyecto fue supervisado por el profesor Maurice Halstead quién trabajó en la decompilación durante los años 1960 y 1970 y publicó técnicas del cual forman las bases para los decompiladores de hoy.

A lo largo de las últimas décadas han proporcionado diferentes usos a los decompiladores. En 1960 los decompiladores fueron usados para ayudar en el proceso de conversión de programas desde la segunda a la tercera generación de computadoras, de esta manera no gastaría "tiempo–consumo" en volver a escribir programas para la tercera generación de máquinas. Durante 1970 y 1980 los decompiladores fueron usados para portabilidad de programas, documentación, recreación de los últimos códigos fuentes perdidos, debugging, y la modificación de los binarios existentes.

A partir de 1990 los decompiladores han emprendido una herramienta de ingeniería en reversa capaz de ayudar al programador con algunas tareas como: verificación de

software para la existencia de código basura, verificando que el compilador genere código correcto, la traducción de programas binarios desde una máquina a otra y entendiendo la implementación de una función de librería particular.

4.2.2 El Proceso de la Decompilación.

Los problemas principales con decompilación son la separación de datos y código (obteniendo un completo desamblaje del programa), la reconstrucción de estructuras de control y el descubrimiento de nivel alto de tipo de datos.

Los pasos principales en un proceso de decompilación basado en un nivel alto (HLL) son:

- Decodificar el formato del archivo binario
- Decodificar las instrucciones máquina en código ensamblador para la máquina.
- Realizar análisis semántico para descubrir algún nivel bajo en la búsqueda de tipos de datos como variables long y a simplificar las instrucciones decodificadas basadas en su semántica.
- Realizar un análisis del flujo de datos para eliminar aspectos de nivel bajo de la representación intermedia que no existe en los niveles altos: registros, códigos de condición, referencias de pilas.
- Realizar un análisis del flujo de control para recuperar las estructuras de control disponibles en cada procedimiento (es decir loops, condiciones y niveles anidados).

- Realizar un análisis de tipos de datos en nivel alto como son los arrays y las estructuras.
- Generara el código de nivel alto desde el código intermedio transformado.

4.2.3 Aplicaciones Específicas Decompiladores.

Hay un número de aplicaciones que generan un código intermedio (bajo nivel), el cual es interpretada por una máquina virtual. Con algunas aplicaciones este código de bajo nivel es puesto dentro de un ejecutable. El código intermedio es el lenguaje ensamblador de la máquina virtual. Algunos ejemplos son:

- MultiRipper: para Windows y Delphi/C++ Builder, programa que extrae archivos dentro de otros archivos. Para aplicaciones Windows extrae los recursos windows y guarda en otro disco y para aplicaciones Delphi/C++ Builder recupera el código y el proyecto de Delphi.
- SourceAgain: decompilador Java. Recupera correctamente el control de las estructuras Java y optimizaciones desde el bytecode. Además este soporte irreducible para gráficos, tipo de conclusión polimórfica, reorganización de paketes y más proporciona un soporte para depurador

4.3 Reingeniería de Programación.

La tecnología de la reingeniería de programación, es un enfoque dirigido a crear sistemas de información en base a componentes, que pueden ser extraídos de sistemas

existentes y reagrupados para producir nuevos sistemas. Resulta interesante que este nuevo enfoque, respecto a las tecnologías de la información va en apoyo de muchas de las ideas que están detrás de la reingeniería.

El cambio, desde una visión funcional a una visión de reproceso plantea nuevos requerimientos con respecto a la forma en que se especifican y diseñan sistemas. Ya no será suficiente con descomponer funcionalmente la lógica de una aplicación, manteniéndola separada de los datos sobre los que actúa. Lo que se necesita es un enfoque en la creación de modelos, que pueda ser utilizado tanto para la conversión o mapping de procesos, y pueden dar lugar a cambios en el modelo de objeto de software.

Para estos procesos de reutilización de código existentes, es de importancia crítica la estructuración y la gestión de todas las variables en las que resulten más útiles y significativas.

Para el éxito de la reingeniería es de importancia crucial el proceso de ingeniería. Muchos procesos comerciales son el resultado de una simple evolución en el tiempo, y no han sido objeto de ingeniería alguna, por lo que con frecuencia surge la confusión entre qué pasos de procesos que existen y cómo se ejecutan esos pasos. Según van apareciendo formas de implementar software, pueden introducirse una a una sin necesidad de producir alteraciones en todo el sistema.

4.3.1 Reingeniería del Software.

La reingeniería del software es la tecnología que surge de aplicar las técnicas de Inteligencia Artificial y matemática sofisticada al análisis automatizado y modificación del código fuente de programas, para abreviarlo y hacerlo más eficiente.

La reingeniería del software está empezando a tomar algunas tareas de programación, particularmente las tareas menos creativas, más repetitivas y las automatiza. Estos programas de reingeniería, escritos en idiomas especialmente diseñados, operan en el código fuente de los programas y realizan una variedad de análisis y modificaciones.

Puesto que la reingeniería es una suma de tareas que requieren por lo general mucho tiempo, éstas se dividen en procesos separados que se llevan a cabo secuencialmente, del cual se presenta lo siguiente:

- **Ingeniería Inversa**: "Es un proceso de recuperación de diseño. Las herramientas de ingeniería inversa extrae información acerca de los datos, arquitectura y diseño de procedimientos de un programa ya existente"[40].
- **Reestructuración del código**: Se analiza el código fuente utilizando una herramienta de reestructuración. Las violaciones de las estructuras de programación estructurada se indican, y entonces se reestructura el código.
- **Reestructuración de datos: "**Un programa que posea una estructura de datos débil será difícil de adaptar y de mejorar. De hecho para muchas aplicaciones,

[40] Ingenieria del Software, Roger S. Pressman, pág. 512

la arquitectura de datos tiene más que ver con la viabilidad a largo plazo del programa que el propio código fuente"[41].

- **Ingeniería Progresiva**: "Las aplicaciones se reconstruyen utilizando <<motor de reingeniería automatizado>>. Se insertaría el viejo programa, que lo analizaría, lo reestructuraría y después regeneraría mejores aspectos de calidad del software."[42]

4.3.2 Desensamblador.

Desensamblar significa el proceso de trasladar un programa ejecutable en su representación equivalente ensamblado. El gran problema desensamblado los programas es determinar que código (instrucciones) y que es dato, como ambos están representados en la misma manera de máquinas actuales.

Tenemos por ejemplo el desamblador llamado Win32 Program Disassembler, el cual tiene una estrategia desamblando en línea en archivos windows de 32 bits ejecutables. El programa trabaja en modo consola (no en interfaz gráfico) usando un comando de línea.

Otra función importante es que este programa decodifica las llamadas a las API's de Win32. No desensambla la sección de los datos cuando esta termina, pero las declaraciones de cadenas están emitidas donde son apropiadamente

[41] Ingenieria del Software, Roger S. Pressman, pág. 512
[42] Ingenieria del Software, Roger S. Pressman, pág. 512

4.3.3 Compatibilidad Binaria.

4.3.3.1 POSIX

POSIX son las iniciales de Portable Operating System Interface, La Interfaz de Portabilidad de Sistemas Operativos, se ha convertido rápidamente en un estándar para proporcionar unos servicios de operatividad mínimos.

Es un estándar que debe cumplir todo sistema operativo que pretenda ser considerado como Unix; este estándar se concreta por un conjunto de especificaciones técnicas que aseguran una evolución previsible en estos sistemas.

Linux es compatible a nivel de código fuente con el estándar POSIX de Unix. Además, también se ajusta al estándar en el que se basan Unix SystemV y BSD. Esta compatibilidad permite que programas desarrollados para una versión de Unix que soporte un mismo estándar, se compilen en Linux y puedan ejecutarse sin modificaciones.

A nivel binario también es compatible con el estándar POSIX, en lo que se refiere a la gestión de procesos. Esta característica está presente, por ejemplo, en el *shell* bash y csh, en la internacionalización de los pseudoterminales y la gestión de consolas virtuales.

4.3.3.2 Extensiones POSIX

Es un estándar que pretende conseguir la portabilidad del software a nivel del código fuente. En otras palabras, un programa escrito para un sistema operatvo que sea compatible con POSIX se compila y ejecuta sobre cualquier otro POSIX aunque sea de otro fabricante distinto.

El estándar POSIX define la interfaz que el sistema operativo debe ofrecer a las aplicaciones; el juego de llamadas al sistema. POSIX está siendo desarrollado por IEEE (Institute of Electrical and Electronic Engineering) y estandarizado por ANSI (American National Standards Institute) e ISO (International Standards Organization). Evidentemente POSIX está basado en Unix. La mayor parte de los S.O. (incluido Windows NT) tienden en sucesivas versiones hacia la compatibilidad POSIX.

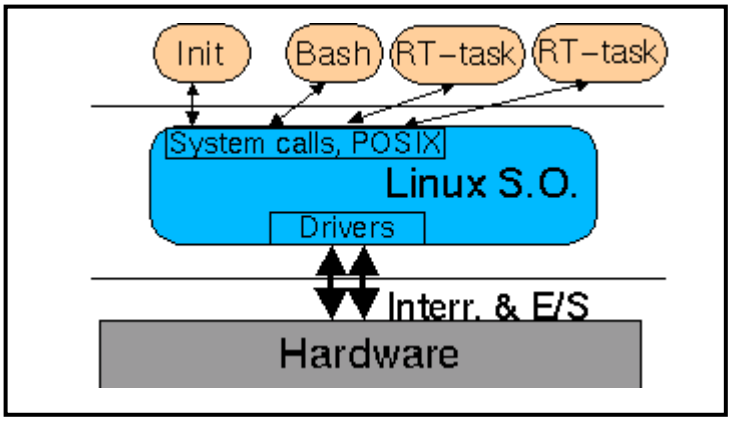


Fig # 4-1. POSIX

El trabajo en la definición del estándar POSIX está dividida en varios grupos de trabajo en los que participan fabricantes de ordenadores, empresas de software, representantes de distintos gobiernos, entre otros. Evidentemente, la mayor parte de estas extensiones están relacionadas con la gestión del tiempo y las prioridades de los procesos, también hay llamadas al sistema para facilitar la comunicación entre procesos.

Las extensiones POSIX están pensadas para tener un mayor control sobre la administración de los recursos del sistema operativo.

4.3.3.3 XPG/SUS

La Single UNIX Specification (una extensión inicial y ahora independientemente mantenida de la colección Guía de Portabilidad de X/Open) especifica un número de facilidades básicas no definidas por POSIX. Actualmente se encuentra en desarrollo para este tipo de portabilidad.

4. 4 Estudio de sistemas de desarrollo de programación que se ejecuten bajo varias plataformas

Hoy en día y, dado el número de Sistemas Operativos que se encuentran en el mercado: Windows en todas sus versiones, Unix, Mac y Linux con todas sus variantes (Suse, Red Hat, Slackware, ...) se está caminando hacia un sistema de programación multiplataforma, de tal forma que, un mismo código pueda ser instalado en múltiples máquinas que corran distintos sistemas operativos.

No obstante, este cambio nos cuenta un gran esfuerzo, ya que se encuentra el entorno de desarrollo que sea lo suficientemente versátil y capaz de llegar a propósitos establecidos. Se trabaja con varios sistemas de desarrollo libre en su versión Linux, como son Delphi (Windows y no libre) y Lazarus (Linux y libre), Visual Basic (Windows y no libre) y Phoenix (Linux y libre), WideStudio (multiplataforma y libre).

4.4.1 Java, Corba y RMI (Remoted Method Invocation)

"Java es un potente lenguaje para programar computadoras que resulta divertido de usar para los principiantes al tiempo que es de gran utilidad para los programadores experimentados que construyen sistemas de información sustanciales"[43].

Java ha resultado ser una bomba en el mundo de la computación, esto se debe a la portabilidad de las aplicaciones de Java que brinda muchas ventajas para trabajar en ambientes multiplataforma.

4.4.2 Java y CORBA

Los modelos de objetos subyacentes a Java y Corba se complementan:

- Ambos modelos soportan el concepto de Interfaces Abstractas.
- Los tipos de datos que se manejan en el IDL corresponden de manera natural a los tipos de datos de Java.

[43] Deitel yDeitel, Como programar en JAVA, pág 2

- Sus mecanismos de herencia son casi idénticos.
- Los módulos de Corba corresponden directamente a los paquetes (packages) de Java.

Además de que ambas tecnologías tienen un modelo de objetos muy compatibles, es decir, Java permite crear objetos portables y los distribuye de manera sencilla mientras que Corba permite interconectar dichos objetos e integrarlos con el resto de los sistemas de cómputo existentes, o bien con aplicaciones de objetos escritas en otros lenguajes.

4.4.3 Java y RMI (Remote Method Invocation)

RMI es nativo de Java, es decir, es una extensión al núcleo del lenguaje. RMI depende totalmente del núcleo de la serialización de objetos de Java, así como de la implementación tanto de la portabilidad como de los mecanismos de carga y descarga de objetos en otros sistemas, etc.

El uso de RMI resulta muy natural para todo aquel programador de Java ya que éste no tiene que aprender una nueva tecnología completamente distinta de aquella con la cual desarrollará. Sin embargo, RMI tiene algunas limitaciones debido a su estrecha integración con Java, la principal de ellas es que esta tecnología no permite la interacción con aplicaciones escritas en otro lenguaje.

RMI como extensión de Java, es una tecnología de programación; fue diseñada para resolver problemas escribiendo y organizando código ejecutable. Así RMI constituye

un punto específico en el espacio de las tecnologías de programación junto con C, C++, etc.

4.4.4 Sistemas multilenguaje

Construir sistemas que involucren varios lenguajes utilizando esta tecnología es una tarea bastante compleja, es necesario construir puentes para ir de un lenguaje a otro. La complejidad de este trabajo depende en gran parte de cuáles son las tecnologías que se pretende unir y esto hace que la complejidad y el tiempo de desarrollo de un sistema completo se incremente significativamente.

Así durante el desarrollo de una aplicación, cada desarrollador puede hacer su parte en el lenguaje que conoce o que va más de acuerdo a una parte del problema.
Una importante consideración, es el sistema operativo y el hardware con el que cuenta la organización, particularmente; si el ambiente de cómputo es heterogéneo, con PCs y sistemas Unix del que requieran tener acceso a la base de datos.

Tras la evaluación, las conclusiones sobre cada herramienta son:

Producto	**Delphi 1.0**	**Power Builder 4.0**	**SQL Windows 5**	**Omnis 7 Ver.3**	**Vision 2**	**Visual Basic 3.0**
Marca	Borland	PowerSoft/Sybase	Gupta	Blyth Software	Unify	Microsoft
Categoría	Low-End Client	Multiplataforma	High-End Client	Multiplataforma	Multiplataforma	Low-End Client
Medio de Distribución	Floppies y CD-ROM	CD-ROM	CD-ROM	CD-ROM	Floppies	Floppies
Plataforma de Desarrollo Mínima	386SX, 6MB, 35 MB HD	386SX, 8MB, 19MB HD	386SX, 8MB 28 MB HD	386DX, 8MB (en plataforma	386DX, 8MB (en plataforma	386SX, 4MB, 10 MB HD

				PC)	PC)	
Plataforma de Desarrollo Recomendada	486, 12MB, 60 MB HD	486, 16MB, 45 MB HD	486, 16MB, 45 MB HD	486, 16MB (en plataforma PC)	486, 16MB (en plataforma PC)	386DX, 8MB, 40MB HD
Plataformas de Implantación Soportadas	Windows 3.x	Windows 3.x, Windows NT, OS/2 , Macintosh, UNIX	Windows 3.x	Unix (Open Look y Motif), Windows y Macintosh	Unix (Open Look y Motif), Windows y Macintosh	Windows 3.x
Utilerías Extra **Incluidas**	Herramientas para desarrollo en grupo. Constructor de consultas visuales. Código fuente de sus componentes	Power Builder Enterprise, Team/PDMS, Desktop y Library for Lotus Notes. Muy completa gama de herramientas	Team Windows (control de trabajo en grupo), herramientas de control y monitoreo, compilador a C, Report Windows	Silver Run Case de dos vías	Desarrolladores 4GL: Accell SQL y Accell IDS	ODBC, controles mejorados y ejemplo. Visual Design Guide y la Microsoft Knowledge Base. Gran cantidad, pobre calidad
Servidor de SQL Local	Interbase SQL solo	Watcom SQL solo	SSQL Base solo	Omnis SQL Client Database	Ninguno, pero es muy accesible al precio del manejador de base de datos Unify 2000, otra opción es Unify RDBMS.	Engine de Access 1.1
Reporteador	Report Smith	InfoMaker	Queso Reporter	Reports y Ad Hoc	Ninguno	Crystal Reports for VB.
Ventajas	Utiliza como lenguaje de programación Object Pascal, que es simple pero poderoso. Código compilado de mucha velocidad de ejecución.	Power Builder es la herramienta profesional más popular del mercado. Recientemente adquirida por Sybase, se puede esperar más integración con él.	Es quizás el producto más fácil de aprender, y de mayor rapidez de desarrollo. Tiene una alianza impor-tante con Microsoft. Es posible generar	Posible migrar la plataforma de ejecución después de hacer la aplicación. Soporta una amplia variedad de formatos para gráficos.	El producto más flexible de los evaluados, es posible generar una aplicación en cualquier plataforma y ejecutarla en cualquier otra interface	Producto pionero y líder en programación general en Windows. Requiere pocos recursos, es rápido y fácil de aprender y

	Diseñado para cliente /servidor		aplicaciones sin escribir código.		consistente e intuitiva	usar. Cantidad ilimitada de Add-Ons
Desventajas	Carece de facilidades para control de versiones o creación de aplicaciones de gran tamaño. Primera versión del producto.	Ambiente de programación jerárquico, un poco diferente al normal. Cambiar de dueño es una incertidumbre.	Requerimientos de hardware sumamente elevados. Su próxima versión usará controles OCX. Su interface es poco intuitiva.	Código interpretado, de bajo desempeño. Altos requerimientos e incompatibilidad con Stacker. No está orientado a PCs.	Su interface choca con el CUA de la plataforma de ejecución. Requerimientos de hardware sumamente elevados.	Servidor. Muy limitado en su diseño y ha quedado obsoleto con el tiempo. Código-P de bajo desempeño.
Observaciones	Una novedosa herramienta, gran velocidad de ejecución, poderoso lenguaje de programación Borland atraviesa una etapa incierta en su futuro, pero aún así ha tenido una gran aceptación.	Quizás la más difundida herramienta del mercado, cuenta con un gran soporte de terceros. Si Sybase sabe integrar su DBMS con ésta, la próxima versión será una excelente opción.	Se perfila para dominar el mercado de herramientas cliente/servidor, gracias a su facilidad de uso. Sin embargo la próxima versión, es una duda al usar controles OCX.	Presentó muchos problemas para instalarse y nunca funcionó a satisfacción. Una PC no es la mejor plataforma para desarrollo ni para ejecución. No recomendable.	Si el problema a resolver implica el uso de varias plataformas de implantación, y el personal no está demasiado acostumbrado a la interface, ésta, parece, es la opción correcta.	Punto de partida y comparación para los demás productos. La versión 4 promete mejoras pero no suficientes para ser competitivo.

Tabla # 4. *Tabla comparativa de Herramientas Visuales.*

4. 5 Ejemplos de software multiplataforma

4.5.1 VISION

Características Generales

Nombre: Vision.

Versión: Release 2.0

Marca: Unify - Visix (Componentes del run-time).

Categoría: Multiplataforma.

Medio de Distribución: Floppies.

Plataformas de Desarrollo Soportadas:

Para Windows 3.x, Procesador 386DX como mínimo, recomendable 486 o superior, mínimo 12 MB en RAM, recomendado 16 MB de RAM y de 34 a 40 MB en disco duro.

También soporta las plataformas Macintosh y Unix.

Ventajas:

- Es muy fácil el desarrollar aplicaciones tipo 'Query by Example' para lo que no es necesario ningún código extra. Es fácil configurar en Windows, nos permite la reutilización de aplicaciones, además cuenta con un Repositorio de Datos. La migración de datos Unix-Windows es sencilla, sólo hay que volver a compilar las aplicaciones desarrolladas.

Desventajas:

- Ocupa demasiados recursos, más que cualquier otra herramienta evaluada (5 MB de RAM mínimo para ejecución, no importando si la aplicación es chica o grande, y de 5 a 7 MB de RAM para cargar el ambiente de desarrollo).
- No cuenta con un Reporteador aunque puede utilizar uno externo.
- No cuenta con capacidades para la creación de gráficos.

- El editor que utiliza es externo. Es muy pobre en este aspecto, debido a que puede prestarse a confusión y a un manejo incorrecto del editor.

Observaciones

Si el desarrollo y producto final van a usarse sobre Unix es una buena opción, pero hay que revisar Omnis 7, Power Builder y SQL Windows (versión para Solaris).

Documentación

Impresa: 9 Manuales, desde conceptos generales que maneja Vision, Integración al DBMS, Diseño de Aplicaciones, Referencia de 4GL, Diseñando una Aplicación, Conexión a Base de Datos, entre otros.

En Línea: Muy buena para explicar los Comandos y Menús utilizados para el diseño de una aplicación pero sin ejemplos de los comandos 4GL, que sólo hay en manuales.

Escalabilidad:

Amplia, gracias a la facilidad de migración. Las aplicaciones inicialmente de Windows pueden ejecutarse en Workstations y Macintosh, así como el cambio a mejores servidores.

Manejo de Gráficos: No tiene.

4.5.2 OMNIS

Características Generales

Nombre: Omnis 7.

Versión: 3.0

Marca: Blyth Software.

Categoría: Multiplataforma.

Medio de Distribución: CD-ROM.

Plataformas de Desarrollo Soportadas*:*

En una PC compatible, configuración mínima procesador 386, 8 MB de memoria y Windows 3.1. Idealmente una 486 con 12 MB de memoria RAM.

Plataformas de Implantación Soportadas:

Unix (Open Look y Motif), Windows 3.x, Windows NT, MacOS, incluye Motorola 68K y Power PC.

Ventajas:

- Permite desarrollar aplicaciones en una plataforma, y posteriormente migrarlas a toda una variedad de plataformas distintas sin necesidad de modificar el código, sólo es necesario compilarlo.
- Proporciona una múltiple variedad de formatos para la elaboración de gráficos; éstos pueden ser escogidos y modificados por el usuario de acuerdo a sus preferencias y necesidades.

Desventajas:

- La gran cantidad de operaciones y beneficios que proporciona Omnis 7, hacen que éste sea una herramienta que consume bastantes recursos físicos para que pueda trabajar adecuadamente.
- No posee interface común de usuario, en la manipulación de ventanas cuando varias están activas al mismo tiempo, algunas teclas iguales tienen un significado diferente para cada ventana.
- Presenta algunas inconsistencias en cuanto al manejo de su ayuda y ésta no es muy completa. No siempre es factible obtener la información deseada de las aplicaciones que se realizan, ya que en ocasiones marca errores irreconocibles.
- Salva automáticamente todas las modificaciones que se realicen, lo cual a veces presenta problemas ya que no siempre desean guardarse todos los cambios que se hacen.

Observaciones:

Su mejor desempeño puede ser sobre equipos Macintoch (debido a que se desarrolló inicialmente sobre esta plataforma), seguido de la versión para estaciones de trabajo Unix.

Facilidad de Migración

Posiblemente la mejor de todas (junto con Vision), al ser capaz no sólo de migrar en servidores, sino también modificar el programa a la plataforma del cliente sin realizar algún cambio.

Escalabilidad:

La capacidad de cambiar la plataforma de hardware del cliente lo hace muy escalable.

Manejo de Gráficos:

Las gráficas de Omnis 7 son de gran calidad, y se podrán seleccionar diversas características para realizarlas. Su presentación es en barras, líneas, y muchas otras, donde se le podrán adicionar diversos atributos como colores, tercera dimensión, rayado en dirección horizontal, vertical con diferentes rangos, etc.

4.5.3 POWERBUILDER.

Características Generales

Nombre: PowerBuilder.

Versión: 4.0

Marca: PowerSoft.

Categoría: Multiplataforma.

Medio de Distribución: Floppies y CD-ROM.

Plataformas de Desarrollo Soportadas:

Procesador 386SX, MS-DOS o PC-DOS versión 3.3 o mayor, 8 MB en RAM, Windows 3.1 o Windows NT. 19 MB de espacio en disco duro. También funciona en plataformas OS/2, Mac y UNIX (Motif y Open Look).

PowerBuilder posee un soporte completo para ambientes Windows de 16 y 32 bits en plataformas Intel, incluyendo Windows 3.1, Windows NT, Win OS/2, Mac y UNIX (Motif y Open Look).

Ventajas

- Permite el fácil desarrollo y distribución de aplicaciones en varias plataformas a nivel corporativo.

Desventajas

- Su ambiente de programación difiere del normal.

Observaciones

Posiblemente es la herramienta de high-end más exitosa del mercado, con muchos desarrollos y herramientas de terceros. Definitivamente sería una mejor opción, pero a la fecha tiene un futuro muy incierto para ser una elección segura.

Facilidad de Migración

Con Watcom SQL como servidor local, es muy fácil desarrollar, para después migrar a otro servidor SQL, ya sea de Watcom u otros.

Escalabilidad

El CODE (Client/Server Open Development Environment) expande la tecnología de los productos de PowerSoft que cubren varios aspectos, como: llamadas remotas a

procedimientos y procesos de transacciones de modelo de datos y pruebas automatizadas.

Manejo de Gráficos

La ingeniería gráfica de PowerBuilder provee de gráficos de segunda y tercera dimensión, de pastel, de barras, columnas, líneas, scatter y gráficas de área.

4.5.4 Resumen de otro Software Multiplataforma

Entre otros ejemplos de software multiplataforma tenemos:

4.5.4.1 CLEAN

Clean es un lenguaje de propósito general, puramente funcional. Es el resultado de varios años de investigación en el Software Technology Research Group de la Universidad de Nijmegen (Holanda).

Plataformas de Desarrollo Soportadas:

Windows, Mac y Linux

4.5.4.2 CLIPPER

Clipper es el nombre de un producto comercial de Computer Associates para Ms-Dos muy utilizado en la época de este sistema operativo (finales de los ochenta y principios de los noventa); Clipper (hoy llamado CA-Clipper) un compilador compatible con

dBASE III+. A este lenguaje y sus distintas evoluciones se le ha seguido llamando Clipper así como Xbase.

Plataformas de Desarrollo Soportadas

para MS-DOS, Windows, OS/2 y GNU/Linux.

4.5.4.3 DYLAN

Dylan (Dynamic Language) es un lenguaje avanzado, dinámico, funcional y orientado a objetos. Permite el desarrollo rápido de aplicaciones, que posteriormente pueden ser optimizadas.

Plataformas de Desarrollo Soportadas

Unix, Macintosh y Windows

4.5.4.4 FORTRAN

FORTRAN (de **FOR***mula* **TRAN***slation*) fue creado por John Backus a mediados de la década de los cincuenta para la IBM, convirtiéndose en el primer lenguaje de programación de alto nivel. Fue pensado para aplicaciones matemáticas y científicas y sigue siendo muy usado en ese terreno.

Los dos estándares principales son Fortran 77 y Fortran 90. El estándar 90 incorporó novedades importantes, como notación de vectores, localización dinámica de memoria, tipos derivados y sobrecarga de operadores, la mayoría de las modernas estructuras de control, etc.

Plataformas de Desarrollo Soportadas

MS-DOS, Unix , Windows, OS/2, Linux

4.5.4.5 KYLIX

Kylix es una herramienta para el desarrollo rápido de aplicaciones nativas Linux de alto rendimiento que simplifica y acelera el proceso de desarrollo mediante la programación visual basada en componentes. Kylix es Delphi y C++Builder para Linux.

Está orientada al desarrollo de aplicaciones de escritorio, aplicaciones para Internet, aplicaciones de bases de datos Cliente/Servidor y aplicaciones distribuidas de n-capas.

La portación de Delphi a Linux es posible gracias a la arquitectura con la que fue diseñado. Los cambios se reducen al núcleo de las llamadas a win32; esta es una ventaja competitiva muy importante frente a las actuales herramientas de desarrollo para Windows.

Tecnologías soportadas

- Bases de datos**:** MySQL, InterBase y otras
- Desarrollo n-capas**:** MIDAS y XML
- Internet**:**
    - XML y DHTML
    - TCP/IP, Sockets, HTTP, FTP, SMTP, POP3 y NNTP
    - Apache API y CGI

La popularidad de Linux como Servidor WEB y Servidor de Correo Electrónico ha generado una gran demanda de aplicaciones para la que será posible ofrecer una respuesta rápida y de alto rendimiento gracias a Kylix.

Plataformas de Desarrollo Soportadas

Linux, Windows.

4.5.4.6 JBUILDER

Borland® JBuilder™ es el ambiente multiplataforma líder en la construcción de aplicaciones Java. JBuilder, hace fácil el desarrollo de EJB, Web, XML y de aplicaciones de bases de datos gracias a sus diseñadores en dos direcciones ("Two-way-tools") y sus características de integración con los servidores de aplicaciones.

Requerimientos mínimos

JBuilder™ 7 Enterprise

- 256 Mb en RAM - (512 Mb recomendados)
- 700 Mb de espacio en disco
- Unidad de CD-ROM
- Monitor de alta resolución (1024x768 o superior a 256 colores)
- Mouse

En Windows®

- Intel® Pentium® II/233 MHz o compatible
- Microsoft® Windows® 2000 (SP2), XP o NT 4.0 (SP6a)

En Linux®

- Intel Pentium II/233 MHz o compatible
- Red Hat® Linux 6.2 ó 7.2 con GNOME o KDE

En Solaris™

- UltraSPARC® II o superior
- Solaris™ 7 (2.7) o Solaris 8 (2.8)

En Apple® Mac® OS X

- Procesador G3 a 350 MHz o superior
- Mac® OS X release 10.1
- (Dado que no hay application servers disponibles en la plataforma Apple Macintosh, algunas características de J2EE no están disponibles)

JBuilder™ 7 SE

- 128 MB en RAM (256 MB en RAM recomendados)
- 650 MB de espacio en disco
- Unidad de CD-ROM
- Monitor de alta resolución (1024x768 o superior a 256 colores)
- Mouse

En Windows®

- Intel® Pentium® II/233 MHz o compatible
- Microsoft® Windows® 2000 (SP2), XP o NT 4.0 (SP6a)

En Linux®

- Intel Pentium II/233 MHz o compatible
- Red Hat® Linux 6.2 ó 7.2 con GNOME o KDE

En Solaris™

- UltraSPARC® II o superior
- Solaris™ 7 (2.7) o Solaris 8 (2.8)

En Apple® Mac® OS X

- Procesador G3 a350 MHz o superior
- Mac® OS X release 10.1

JBuilder™ 7 Personal

- 128 MB en RAM (256 MB en RAM recomendados)
- 500 MB de espacio en disco
- Unidad de CD-ROM
- Monitor de alta resolución (1024x768 o superior a 256 colores)
- Mouse

En Windows®

- Intel® Pentium® II/233 MHz o compatible
- Microsoft® Windows® 2000 (SP2), XP o NT 4.0 (SP6a)

En Linux®

- Intel Pentium II/233 MHz o compatible
- Red Hat® Linux 6.2 ó 7.2 con GNOME o KDE

En Solaris™

- UltraSPARC® II o superior
- Solaris™ 7 (2.7) o Solaris 8 (2.8)

En Apple® Mac® OS X

- Procesador G3 a 350 MHz o superior
- Mac® OS X release 10.1

Principales características

Productividad

- Soporte a los más recientes estándares de Java, incluyendo el soporte al JDK 1.4
- Ambiente de alta productividad que integra UML, editores, unidades de prueba, depurador, herramientas para JavaDoc, refactoring, administrador de configuraciones, codeinsight, ayuda en línea y mucho más.
- Desarrollo visual con herramientas en dos direcciones ("Two-Way-Tools") para la construcción de aplicaciones mediante componentes de "Arrastrar y soltar".
- Soporte al diseñador de formas de IBM VisualAge for Java, para continuar los desarrollos que se hayan iniciado con dicha herramienta.

Depuración

- Depurador gráfico basado en JPDA (Java Platform Debugging Architecture).
- Depuración avanzada, incluye depuración de JSP, depuración local o remota y más

Desarrollo en Equipo

- Desarrollo en equipo integrado, mediante el soporte a herramientas de administración de concurrencia del código fuente como Rational ClearCase, Microsoft Visual SourceSafe y el Concurrent Version System (CVS).

XML y el desarrollo con Java

- Incluye las herramientas y características para soportar XML que permiten a las aplicaciones: presentar, transformar y validar documentos XML, Data binding y la manipulación programática de documentos XML y la transferencia de datos entre XML y las bases de datos.

- Soporte a parsers XML.

Desarrollo Web

- Desarrollo Web integrado para creación de JSP o servlet.
- Soporte integrado a Tomcat 3.2 y 4.0
- Soporte a servidores Web externos como Borland Enterprise Server 5, BEA WebLogic 5.x, 6.x y 7, WebSphere 4 e iPlanet 6.
- InternetBean Express, desarrollo basado en componentes visuales con acceso a bases de datos y a Enterprise JavaBean
- Distribución Web mediante la creación automática de archivos WAR

4.6 Caso de Estudio 1: Programación de las API's de Windows en Linux

Con un sistema operativo sencillo y antiguo, la programación es directa, el programador controla el código, y recursos (disco duro, memoria, pantalla etc.). Con los sistemas actuales esto no es posible, no esta permitido. La API (Aplication Program Interface) es el intermediario que emplean los sistemas operativos actuales para el efecto.

Una API es un método específico prescrito previamente para ser usado por otros programas. Es una especie de puente, de puerta de entrada, que facilita la comunicación entre los programas y los recursos del sistema. Mediante la API de un sistema operativo los programadores pueden efectuar peticiones al sistema operativo o a otra aplicación.

La API en un sistema operativo es la razón por la que un software diseñado para Mac OS es incompatible con Windows, de hecho; los populares emuladores convierten las llamadas API de un programa en llamadas al API del sistema operativo existente, actúan por tanto como un traductor de APIs.

En programación de alto nivel, un programa no siempre ejecuta las tareas por si mismo, puede llamar a otros programas que lo hagan. En el caso de los sistemas operativos, los programas frecuentemente delegan en él varias tareas. Por ejemplo, los programas que corren bajo Windows no tienen más que especificar al sistema operativo que tipo de ventanas quiere y él se encargará de generarlas. Por supuesto, la comunicación entre ambos se lleva a cabo con la intermediación de la API.

Las ventajas son evidentes, de esta manera los programadores no tienen que preocuparse de las labores rutinarias y repetitivas, ellos ahorran tiempo y el programa ocupa menos espacio; los programas pueden pasar fácilmente de un sistema a otro (portabilidad) y funcionan perfectamente en cualquier máquina que tenga ese sistema operativo.

En Unix y en Linux no hay ningún problema, la API es siempre compatible y el código es válido. Pero la API de Windows es otra cosa. Al ser propiedad de Microsoft la única manera que tiene un programador de escribir programas profesionales es pagar una cantidad de dinero impuesta por la empresa de Redmond al tener acceso a su API.

Pero la API de Windows, cuyas diferentes funciones se localizan en las numerosas DLLs del sistema (Dynamic Link Library), no funcionan muy bien, están programadas en C++, por lo que pueden surgir conflictos con los programas que son creados con

otros lenguajes de programación, tienden a fallar con cierta frecuencia generando fallos de protección general. En conclusión, los programas son inestables y se obliga a las empresas a invertir en desarrollo más de lo normal.

Los programas de casa tienen todas las ventajas, si algo no funciona, lo cambian. Es decir, los programas de Microsoft en determinadas ocasiones como en la instalación de nuevas versiones, alteran las librerías dinámicas en su provecho, de manera que se establece un buen funcionamiento; lo malo es que pueden interferir con otros programas menos afortunados que los estaban empleando y pueden dejar de funcionar.

En observación, cualquiera que intente modificar la API de Windows, o simplemente descubrir como funciona utilizando ingeniería inversa, es más que probable que se encuentre con una demanda de Microsoft por infringir su patente.

4.6.1 Llamadas a las Funciones del API de Windows

Un programador de aplicaciones Windows además de conocer su entorno de trabajo, debe conocer también su entorno de programación, conocido generalmente como interfaz de programación de aplicaciones de Windows (Windows Application Programming Interface, abreviadamente Windows API).

La característica primaria de la API de Windows son las funciones y los mensajes internos / externos. Las funciones Windows son el corazón de las aplicaciones. Hay más de 600 funciones de Windows dispuestas para ser llamadas por cualquier lenguaje, como C o Visual Basic.

A través de estos mecanismos, se puede realizar gran cantidad de funciones. La utilización de ésta serie de librerías que contienen las funciones de la API pueden solucionar gran cantidad de problemas en la programación, aunque también no debemos desestimar el gran poder destructivo de las mismas.

La depuración de un programa es muy, diferente si éste tiene incorporadas llamadas a APIs, ya que el resultado de las mismas en algunos casos puede llegar a ser incomprensible. Cuando se trabaja con éstas funciones siempre es recomendable tener una buena guía o bien conocer alguien que sepa, para no encontrarnos con posibles problemas.

Las dos principales ventajas que obtenemos con la utilización de APIs, es la gran funcionalidad que se puede dar a una aplicación, y en segundo lugar la gran velocidad de proceso, ya que a menudo es mucho más rápido realizar una función a través de la API adecuada que por medio del lenguaje en si mismo.

Los mensajes son utilizados por Windows para permitir que dos o más aplicaciones se comuniquen entre sí y con el propio sistema. Se dice que las aplicaciones Windows son conducidas por mensajes o sucesos. Conocer todas las APIs de Windows es imposible, ya que tiene gran similitud con aprenderse la guía de teléfonos.

4.6.2 Resumen de algunas funciones de API.

Algunas de las funciones del API, se pueden agrupar en:

- **Funciones de direccionamiento de memoria a 32 bits**. Recupera el API para aplicaciones que utilizan 32 bits.

    - **Funciones de ejecución de aplicaciones.** Tiene la función de cargar y ejecutar aplicaciones. ej: Winhelp
    - **Funciones bitmap.** Crea y maneja los bitmaps.
    - **Funciones callback o rellamada.** Recupera información sobre algo que se está ejecutando, con la posibilidad de hacer algún cambio.
    - **Funciones de cuadro de diálogo común.** Funciones que actúan sobre una caja de diálogo común, cambiando su color, texto, etc.
    - **Funciones de comunicación.** Funciones que gestionan los dispositivos de comunicaciones.
- **Funciones de cursor.** Gestionan todo lo relacionado con el cursor.
- **Funciones DDE (Data Dinamic Exchange).** Gestiona el intercambio de datos dinámico.
- **Funciones de error.** Funciones que gestionan los errores de los dispositivos.
- **Funciones de movimiento y visualización.** Funciones que gestionan los manejos de ventanas y aspecto gráfico.
- **Funciones para las propiedades.** Informa y modifica el estado de las propiedades.
- **Funciones de paleta de colores.**
- **Funciones de dibujo y pintura.**
- **Grupo de funciones OLE.** Funciones que se basan en el manejo de objetos.
- **Funciones Toolhelp.** Normalmente devuelven información sobre los objetos que hay en memoria.
- **Funciones Drag & Drop.** Funciones de arratrar y soltar. Tiene información sobre la posición de arrastre de algún elemento, si se puede arrastrar, etc.

- **Funciones de controladores instalables.** Hacen una gestión de los controladores instalables.
- **Funciones de decodificación LEMPEL-ZIV.** Hacen una gestión de los ficheros comprimidos.
- **Funciones para la impresión.** Devuelve tamaños de letra, página, etc. y activa o desactiva algunas funciones, configura colores, etc. a la hora de imprimir.
- **Funciones SHELL (entorno).** Son las funciones que controlan el entorno.
- **Funciones "Stress".** Controla espacio libre y manejadores de ficheros (handles).
- **Funciones TrueType.** Funciones que controlan los fuentes o tipos de letra a utilizar.
- **Funciones de versión.** Función que controla la versión de los ficheros.
- **Funciones GDI.** Controla todo lo que se refiere a los gráficos, ya sea tipos de letra, manejo de trabajo a la hora de una impresión, prepara la impresora para aceptar datos, etc.
- **Funciones KERNEL.** Lleva a cabo una E/S a ficheros, control de errores, etc.
- **Funciones de usuario.** Son funciones en las que el usuario realiza alguna acción, como activar o desactivar flechas de scrollbar (texto horizontal-vetical), recupera información de mensajes de hardware, etc.

Gran parte de las funciones que las API's ejecutan esta disponibles también mediante mandatos. En resumen, se agrupa las API's según sus funciones como:

- API's para recuperación de valores del sistema.
- API's que devuelven listas de información sobre algo del sistema.

- API’s que permiten la manipulación (crear, cambiar o eliminar) de un determinado tipo objeto.

- API’s, que permiten múltiples funciones en el sistema (enviar mensajes, edición numérica, cálculos matemáticos, etc)
- Cada sistema operativo hace ciertas funciones disponibles para ser usado por programas que ejecutan bajo cierto orden de abrir archivos o aplicaciones.

4.6.3 SWT: Standard Window Toolkit

Este proyecto pretende ser un framework para construir IDE's (Integrated Development Environment) para distintos tipos de herramientas, basado en una arquitectura de plugins.

El toolkit SWT surge como necesidad de cubrir los problemas, a nivel de diseño fundamental, en cuanto a consistencia de interfaz se refiere. El objetivo de SWT es crear un toolkit que sea nativo y a la vez portable. Estas dos características que parecen ser imposibles de cumplir en forma conjunta, son posibles gracias a la API JNI: Java Native Interface. JNI permite que una aplicación Java interactúe con código nativo de la plataforma, por ejemplo bibliotecs .so en Linux.

SWT es una API que representa a un toolkit multiplataforma, en donde encuentra toda la funcionalidad típica de los toolkits. Desde el punto de vista de la implementación, el

SWT es una capa muy delgada entre Java y las librerías nativas del toolkit de la plataforma.

SWT siempre trata de hacer un mapeo uno a uno entre el toolkit nativo y el abstracto. Cuando una característica no está disponible en el toolkit nativo, SWT lo emula. Existen varios ports de SWT para las plataformas más populares como:

- Linux/GTK+ 2.x
- Linux/Motif
- Windows/MFC
- Solaris, AIX, HP-UX / Motif
- QNX/Photon

SWT ocupa muy pocos recursos, es notablemente más veloz y se integra perfectamente con el toolkit nativo. Podemos tener, por ejemplo, una aplicación Java corriendo en Linux con una interfaz gráfica GTK+. Esta misma aplicación, sin cambios puede correr en un entorno Windows pero usará la interfaz nativa Win32.

4.7 Caso de Estudio 2: Análisis de programas que emulan en DOS bajo Linux

4.7.1 CPCEMU

Es un emulador para DOS, que emula un Amstrad CPC 464, 664 o 6128 con muchas extensiones.

Los requerimientos para la instalación son:

- PC AT 386 con gráficos VGA.
- MS-DOS 5.0 o mayor
- Novell DOS (DRDOS) o
- Windows 95 (o 3.1) con salida a DOS
- Tener los archivos pkunzip, zip para la extracción o empaquetado

Hay que establecer el funcionamiento del teclado para sus funciones en lo que el CPCEMU tiene ya implementado en sus propios archivos ROM, al igual que sus menus ya establecidos para el manejo del software.

4.7.2 BOCHS

Es un emulador altamente portable en código abierto (x86) para PC, escrito en C++ que se ejecuta en las plataformas más populares. Incluye una emulación de un Intel CPU (x86), dispositivos de entrada y salida, y una bios.

Actualmente, bochs pueden compilarse para emular un CPU 386, 486 o Pentium. Bochs es capaz de funcionar en la mayoría de los sistemas operativos dentro de la emulación incluso Linux, Windows® 95, DOS, y recientemente Windows® NT 4. Bochs fue escrito por Kevin Lawton y fue mantenido actualmente por este proyecto.

Bochs, puede ser compilado y usado en una variedad de modos, algunos en el que todavía está en desarrollo. El uso típico de bochs es proporcionar una emulación de PC

incluyendo un procesador x86, dispositivos de hardware y memoria. Esto permite ejecutar un sistema operativo y software dentro del emulador en su estación de trabajo, en otras palabras tener una máquina virtual dentro de otra máquina física. Interpreta todas instrucciones power-up to reboot, keyboard, mouse, VGA card/monitor, disks, timer chips, network card, etc,

Bochs fue escrito por Kevin Lawton comenzando en 1994, empezó como un producto comercial. Finalmente en marzo del 2000, la compañía Mandrakesoft, compró el código fuente bajo la licencia GNU – LGPL. En marzo del 2001, Kevin ayudado por pocos desarrolladores movieron toda la actividad de Bochs desde bochs.com a un nuevo sitio bochs.sourceforge.net. La Virtualización de la máquina virtual de bochs permite porciones grandes de simulación para tener velocidad en la ejecución del hardware nativo.

4.7.2.1 Plataformas Soportadas.

Las plataformas soportadas por bochs son las siguientes:

Unix/X11	X windows siempre a estado bien soportada porque fue desarrollado por Kevin Lawton en la plataforma del desarrollo principal. Bryce Denney mantiene la plataforma Unix/ X11 ahora. La mayoría de características y arreglos (no todo) se prueba primero en Unix y entonces portadas a los otros.
Win32	Este software fue terminado por David Ross y se mantiene ahora por Don Becker. Se puede compilar con MS C++ Visual 5.0 o 6.0, o Cygwin[a].

BeOS	Kevin Lawton escribió este código, originalmente a R3/PPC que usa CodeWarrior. Ahora trabaja en R4/x86 con el egcs. Simon Huet escogió el mantenimiento/reutilizando el código GUI de BeOS. En 2001 de septiembre, Bernd Korz de ,of Yellow Tab, Inc.(www.yellowtab.com), tomó el código de BeOS/Zeta.
MacOS X	Emmanuel Mailliard puso el código de Macintosh a MacOS X Carbon API. Ha estado manteniendo la portabilidad de MacOS X desde marzo 2002.
Macintosh	David Batterham drbatter@socs.uts.edu.au o drbatter@yahoo.com portaron bochs al la Mac. Él compiló con CodeWarrior Pro R1 (CW12). Para compilar, tiene que construir cabeceras y Makefiles en una máquina de Unix usando"configure--with-macos".
Amiga MorphOS	Este software esta escrito y mantenido por Nicholai Benalal.
OS/2	No ha sido actualizado en más de un año, y la página de Nick Behnken parece ser inactiva. También, Craig Ballantyne ha portado bochs a OS/2, pero su página web ha desaparecido y su código no ha sido actualizado marzo del 2000. Sus últimas fuentes están en http://www.os2world.com/emulator/computer.html.
Notas: Cygwin es un entorno libre Unix-like para Windows escrito por Steve Chamberlain y ahora mantenido by RedHat. Descargar desde --www.cygwin.com	

Tabla # 5. *Plataformas soportadas por Bochs.*

4.8 Caso de Estudio 3: Análisis de la migración hacia Linux de aplicaciones realizadas en Unix, Win 9x / Win NT y DOS como:

- **Win 9x : Fox Pro 3.0, ejemplos de JAVA y Front Page**
- **Win NT: Sybase Adaptive Server, Explorador NT**
- **DOS: Qbasic, C.**
- **UNIX: Start Office y Corel WordPerfect.**

4.8.1 Windows - Linux

El realizar la ejecución de otras aplicaciones desarrolladas en Windows, Unix hacia Linux, se deben tener en cuenta aspectos como los escritos anteriormente. Las diversas empresas desarrolladoras de software tienen muchos aspectos en las configuraciones de hardware y software que se deben tener en cuenta ya que cada una de estas empresas emprenden tácticas empresariales de competencia con sus aplicaciones.

Para que se pueda ir migrando hacia Linux en el escritorio, se deben superar trabas que son de carácter operacional. Principalmente son:

- Conectividad con otros miembros de la red, a nivel de servicios
- Disponibilidad de aplicaciones nativas de uso general
- Disponibilidad de aplicaciones específicas comerciales o no

Hay servidores/clientes disponibles para la gran mayoría de servicios standard. Y también hay servidores y clientes para servicios propietarios como SMB, usados intensamente en redes Windows.

4.8.1.1 VMWARE

Este primer caso nos emprendemos a ejecutar a Windows dentro de Linux, con el software de máquina virtual llamado VMware, el que podemos cargarlo a todo el sistema operativo Windows y ejecutar todas las aplicaciones instaladas físicamente.

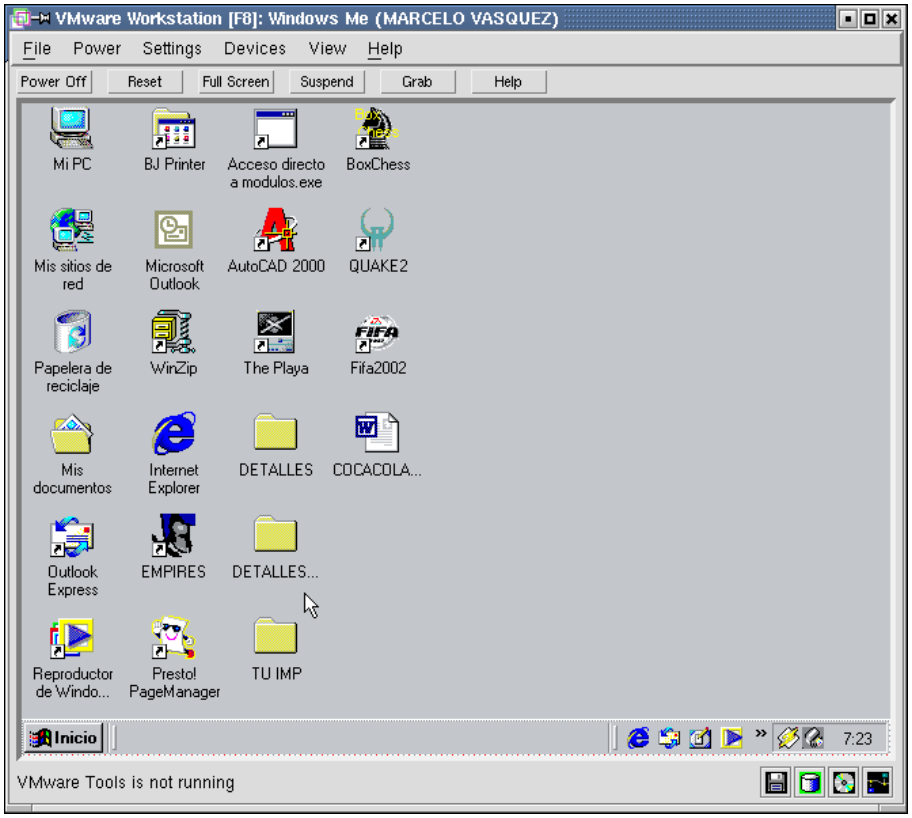


Fig # 4-2. *Presentación de WindowsMe ejectandose a través de Vmware*

Este software permite la integración de diversos sistemas operativos como: DOS 6.0, Windows 3.1 / 95 / 98 / NT / 2000 y XP, así también con las diferentes versiones de Linux como Red Hat, Suse, Caldera entre otros.

Para ejecutar este software de máquina virtual los requerimientos mínimos son los siguientes:

- PC Pentiun II, de 400 Mhz de velocidad
- Memoria mínimo 128 Mb (recomendado 256 MB)
- Tarjeta de video que soporte 256 colores mínimo a 8 bits
- 20 Mb de disco duro para una instalación básica

4.8.1.1.1 Instalación.

Se procede a instalarlo de la siguiente manera:

- El usuario debe ser root
- Copiar el paquete vmware (.rpm) a un directorio temporal
- Realizar un click y verificar su instalación con el administrador de paquetes de Linux (Kpackage).
- Y se procede a instalarlo
- Ejecutar el script de configuración vmware-config.pl
    - Dentro de este script se deben contestar las preguntas que establecen la configuración de samba para la red virtual como se presenta a continuación:
    - Do you want this script to automatically configure your system to allow your virtual machines to access the host file system?
    - Si se está ejecutando samba en el PC, conteste no

  - Si samba no esta ejecutandose en el PC, conteste Yes, y el mismo script realiza las instalaciones de configuración automáticamente, e ingresando un user name y un password con el que se establece la configuración de Samba.
  - Para habilitar el host-only networking responder Yes. Y aparecen las siguientes líneas:
  - Do you want your virtual machines to be able to use the host´s network resourses?
  - Do you want to be able to use host-only networking in your virtual machines?
  - Do you want this script to probe for an unused private subnet?
  - Establecido estos pasos, la configuración del programa presenta el mensaje de configuración completada.
- Una vez configurado ejecutar en la línea de comandos vmware.

En vmware se debe tener en cuenta el sistema operativo del que se va a ejecutar, con la ayuda del wizard se completa la carga del S.O como presentan las siguientes pantallas:

1. Ejecutar el Wizard

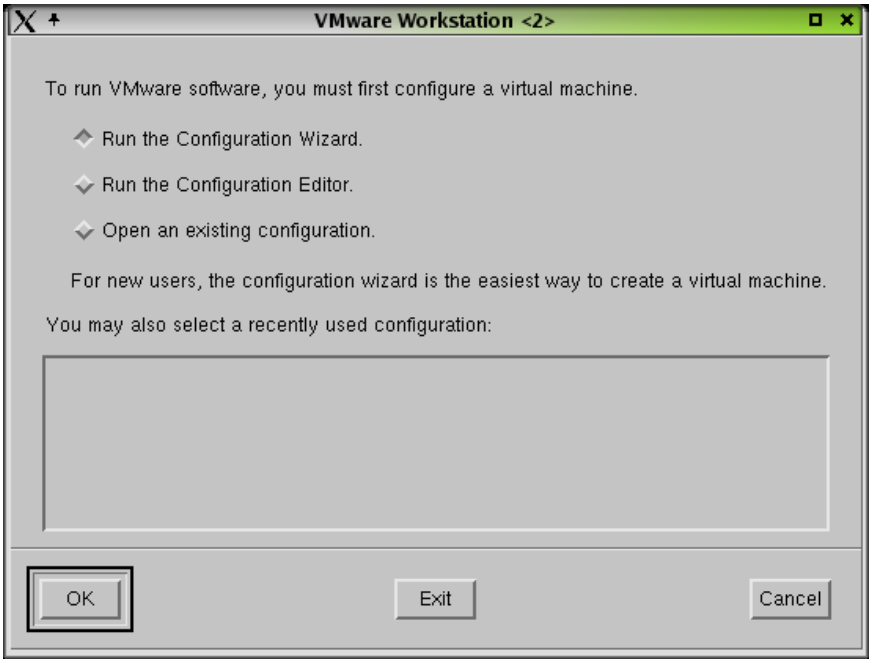


Fig # 4-3. *Wizard de Configuración de Vmware*

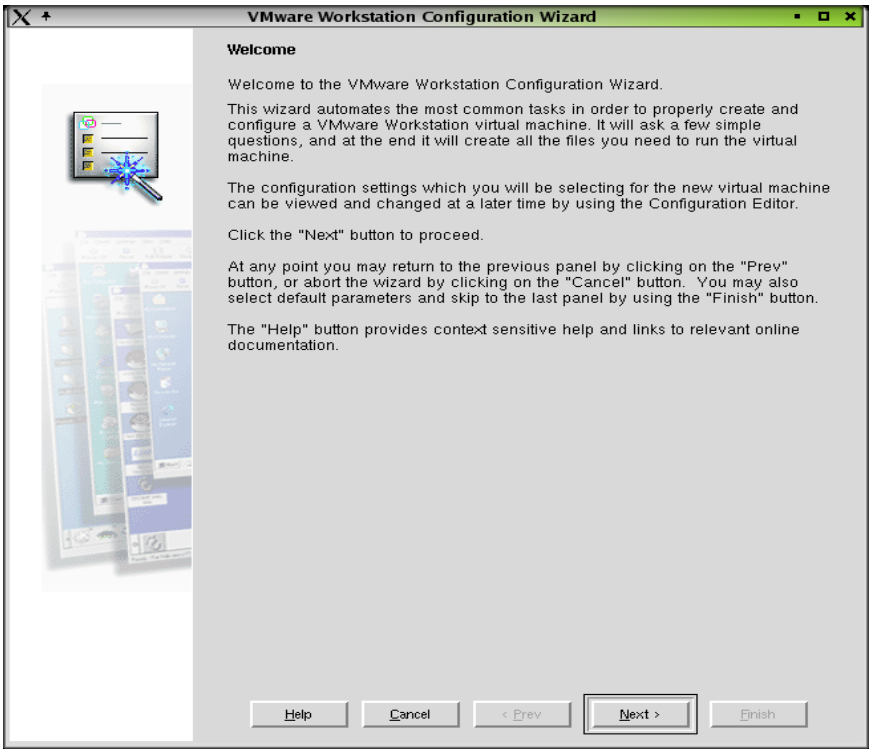


Fig # 4-4. *Wizard de Configuración Inicial de Vmware*

2. Seleccionar el primer ítem

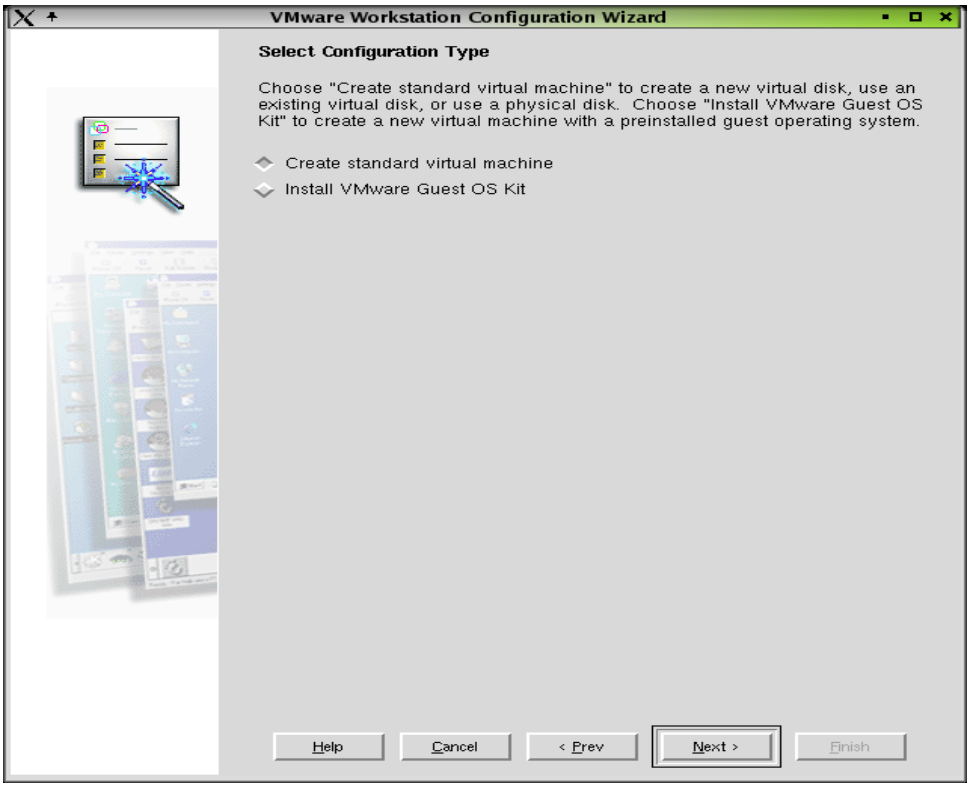


Fig # 4-5. *Wizard para crear la Máquina Virtual*

3. Escoger el S.O a cargar, para el presente trabajo elegimos Windows Me.

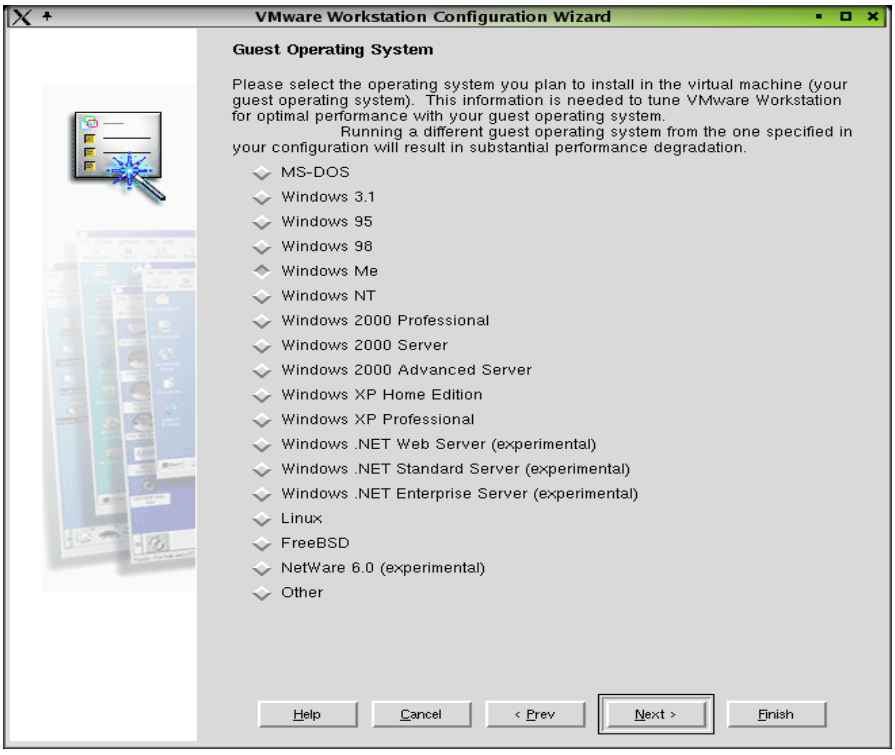


Fig # 4-6. *Selección del Sistema Operativo a Ejecutarse*

4. Escribir el Nombre y el directorio del S.O

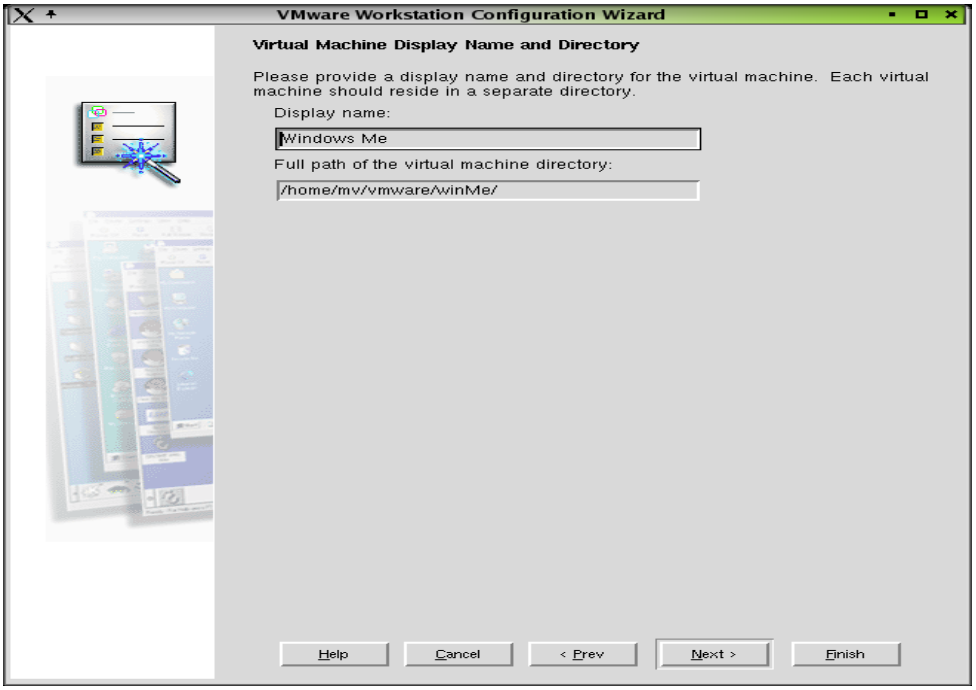


Fig # 4-7. *Nombre y Directorio del sistema Operativo Escogido*

4. En esta pantalla se presentan la configuración de los discos, en el que: si se desea instalar un sistema operativo invitado desde CD-Rom, escogemos la primera opción asignando un disco virtual para el almacenamiento del S.O. En la opción dos sirve para cargar un disco virtual existente o ya creado anteriormente. Y la opción tres se utiliza la partición física del disco donde se encuentra instalado el S.O físicamente. Se ha escogido la opción tres para la demostración.

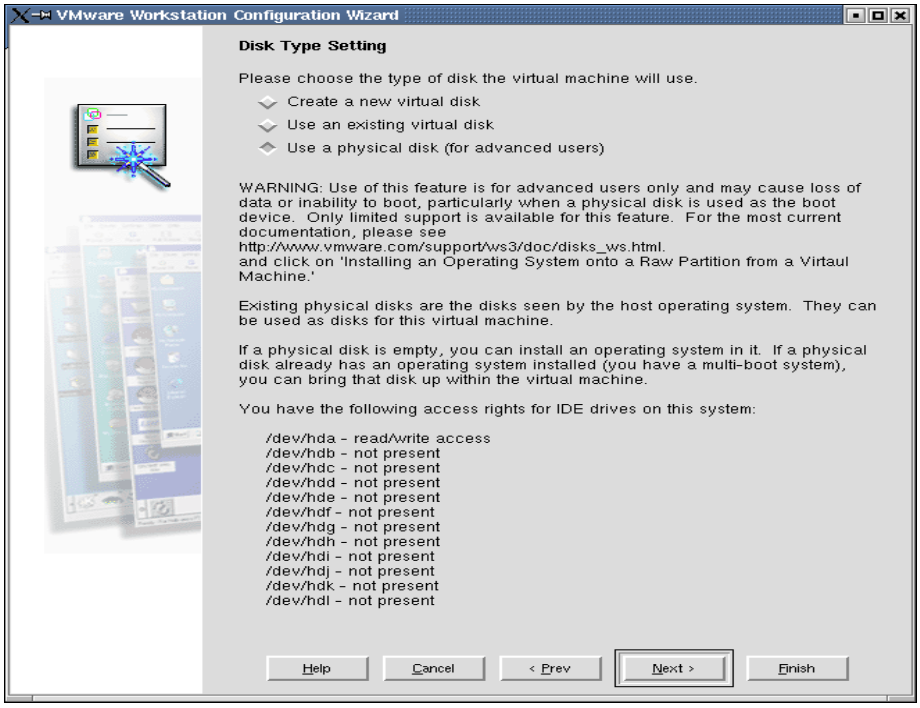


Fig # 4-8. *Tipo de Disco a ser usado dentro de Vmware*

6. Si deseamos utilizar el CD-Rom se habilitaría o no.

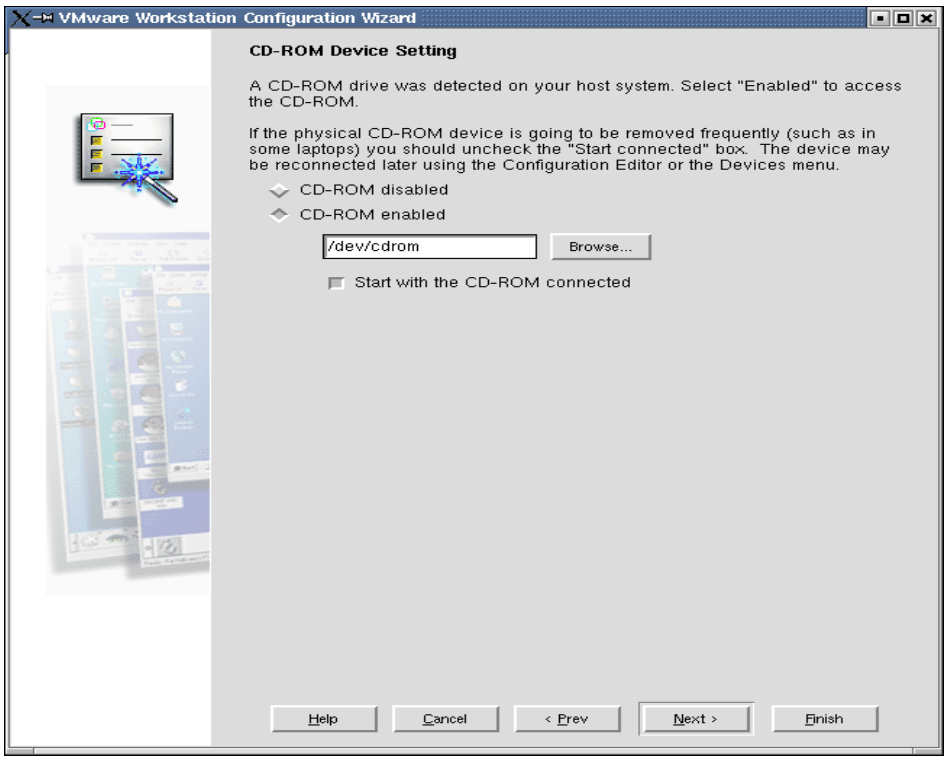


Fig # 4-9. *Habilitar o Deshabilitar el CDROM*

7. De igual manera se habilitaría o no el floppy.

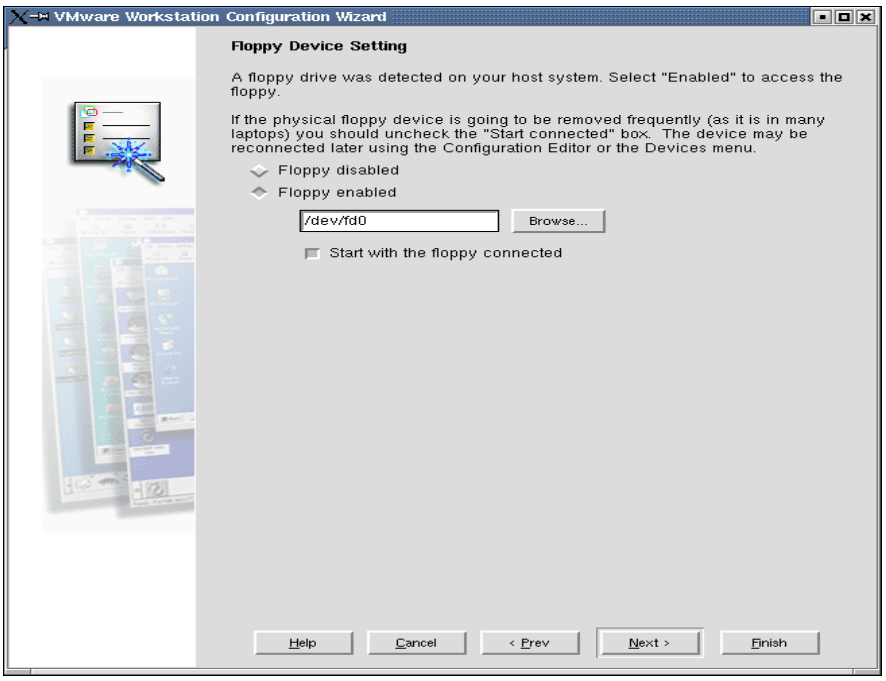


Fig # 4-10. *Habilitar o Deshabilitar el Floppy*

8. Si se forma parte de la red habilitamos las diferentes opciones que se presentan.

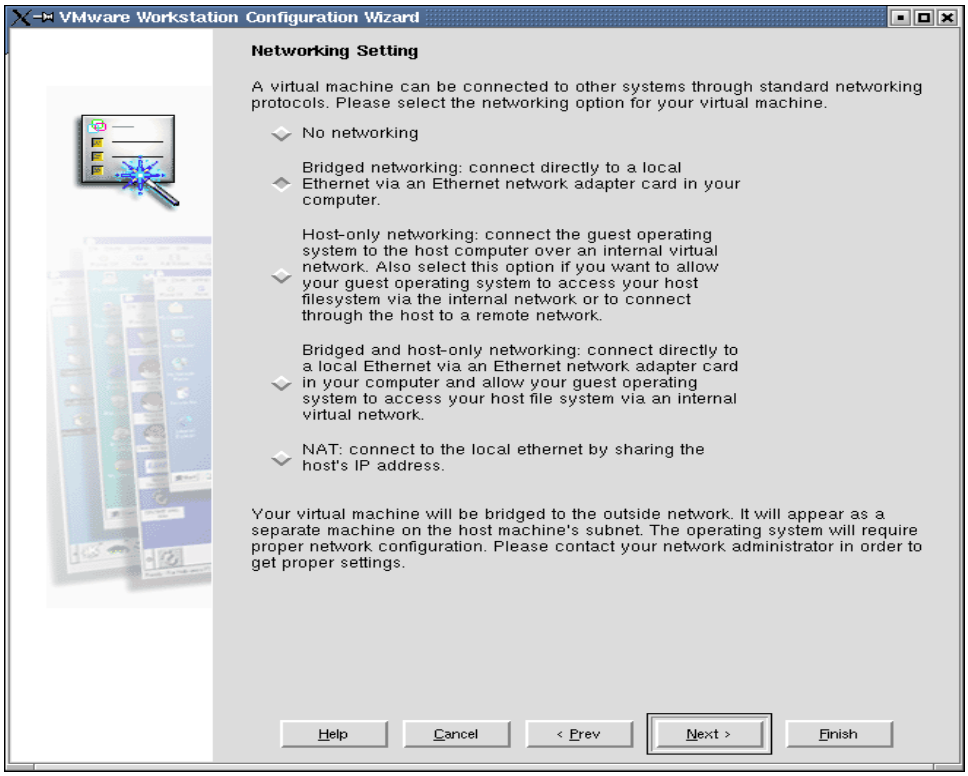


Fig # 4-11. *Escoger el tipo de Networking Protocols*

9. Confirmación de todas las opciones anteriormente detalladas y finalizar.

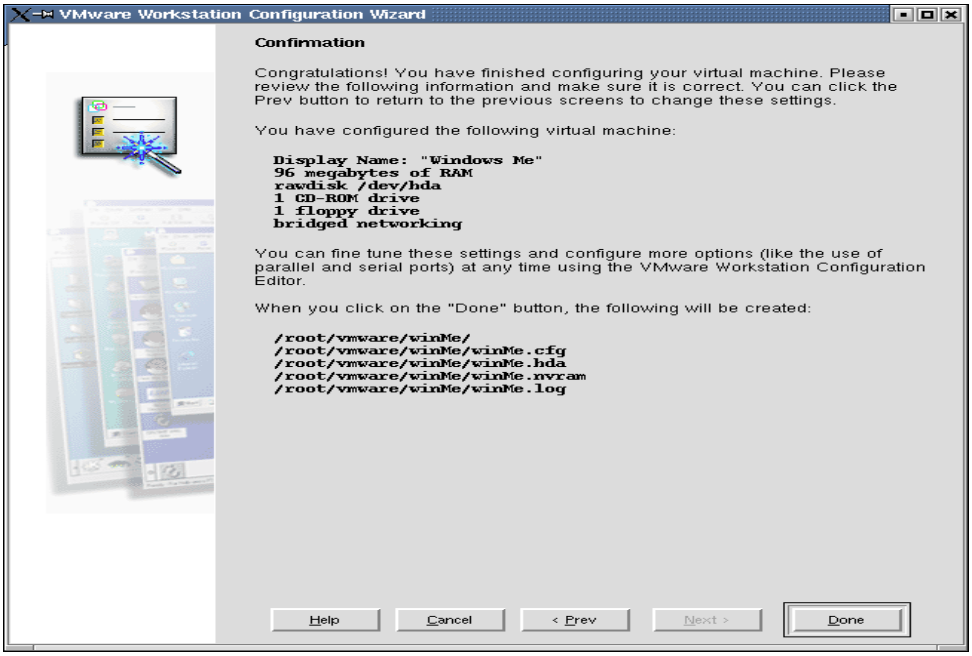


Fig # 4-12. *Confirmación sobre la Máquina Virtual creada.*

10. Presionar Power On para iniciar la máquina virtual configurada.

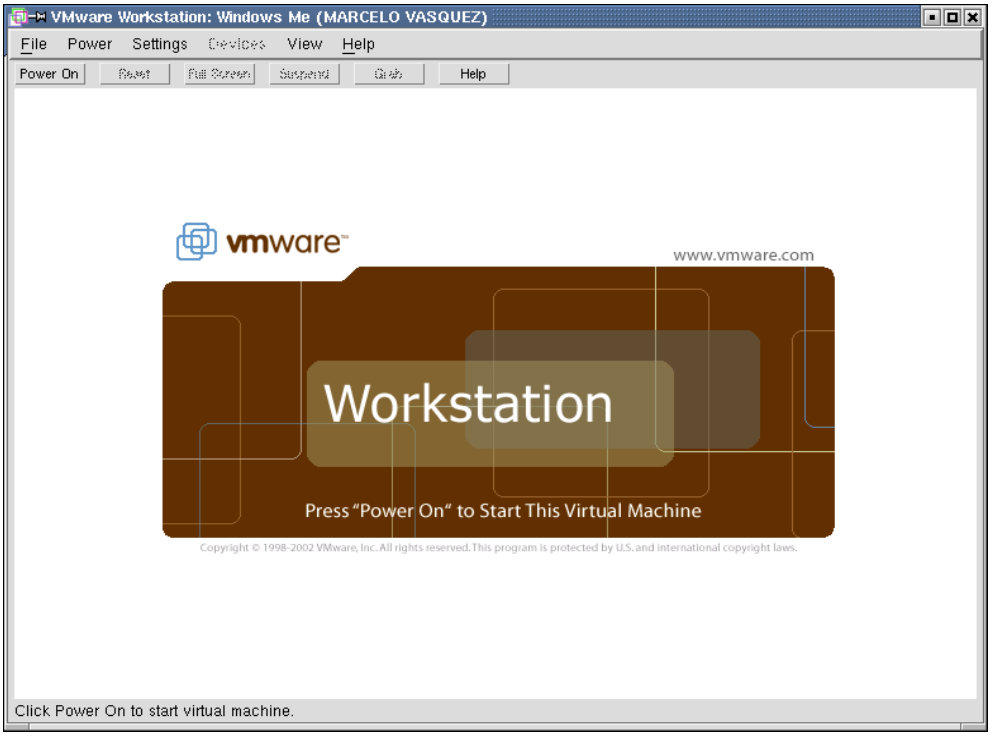


Fig # 4-13. *Pantalla Inicial de Vmware listo para ejecutar el Sistema Operativo seleccionado.*

4.8.1.2 WINE

Para wine se ha utilizado la versión 20021031, tanto para el binario como para el fuente, la instalación del binario nos permite ejecutar pocas aplicaciones, mientras al compilar el fuente e instalarlo esta versión es más estable y se puede ejecutar más aplicaciones.

En la compilación wine reestructura su conjunto de directorios y librerías de enlace con respecto a la versión del Linux con lo que permite la ejecución de más aplicaciones. Volviendo a recompilar los fuentes de wine, se ha ejecutado la aplicación Front Page sin ningún problema en el que se puede trabajar con la mayoría de funciones incluidas en la aplicación; mientras que, con la versión binaria presenta errores en la ejecución y muestra la pantalla de depuración para realizar seguimientos y cambios en la realización de esta aplicación.

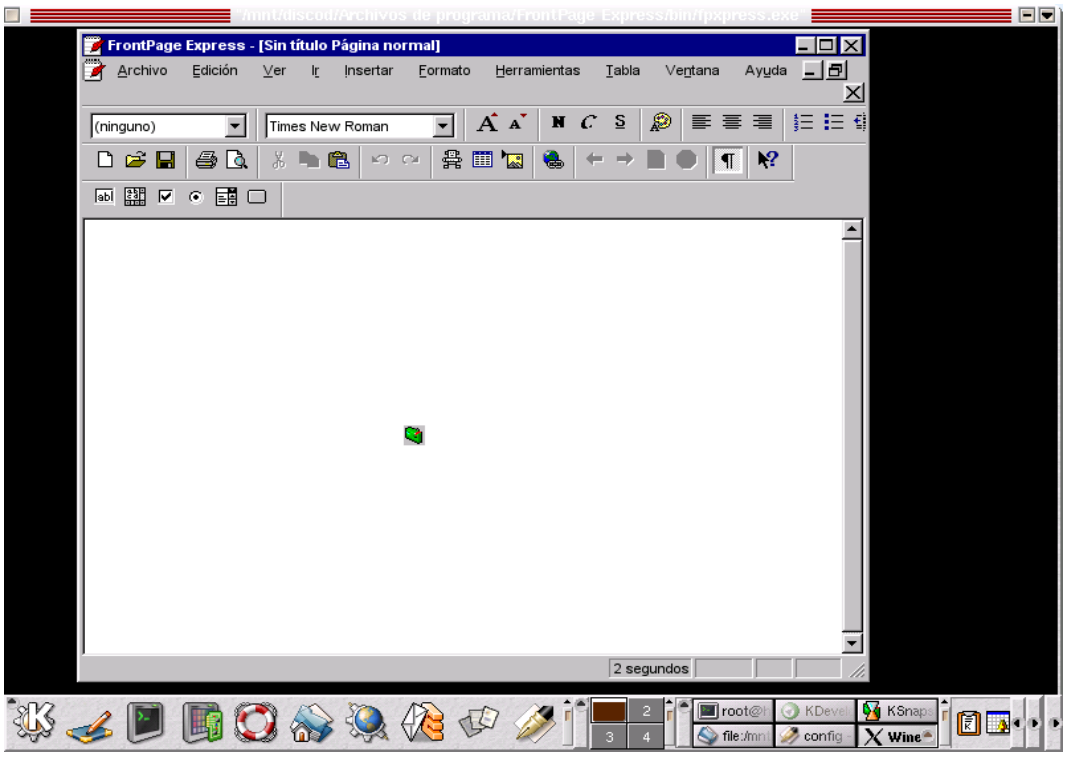


Fig # 4-14. *Ejecución de Front Page con recompilación del Código Nativo de Wine*

Con estas demostraciones, al realizar ciertas comparaciones entre Windows y Linux establecen puntos generales como:

- Estabilidad: quizás sea lo más discutido. En un PC de sobremesa Linux en sus diferentes distribuciones puede que sea más inestable que Windows 2000 y que existe una cierta igualdad en este aspecto. Aceptar el hecho que debido a que el GUI (interfaz gráfico de usuario) no forma parte del núcleo de Linux y sí del de Windows en caso de bloqueo de este componente en Linux matando el servidor X el problema está resuelto. En Windows la situación es distinta y tenemos el típico pantallazo que puede requerir reiniciar.

Pero se propone a modo de pregunta: ¿Porqué siempre que se menciona esta ventaja no sé dice que tener el GUI en el núcleo aporta ventajas de velocidad?; ciertamente debido a las implementaciones de calidad de ambos sistemas operativos que a efectos finales tanto la velocidad como la estabilidad del GUI son relativamente similares. En cuanto a Windows 95/98/ME pues en este caso la ventaja de estabilidad de Linux es más consistente.

- Utilización de Recursos y Rendimiento: la utilización de recursos de Linux es mucho menor que en las distintas versiones de Windows. El rendimiento y capacidades que proporciona Linux sobre máquinas antiguas son superiores a Windows. Con la aparición del Pentium IV, ha quedado en evidencia la importancia de los servidores de altas densidades. Linux es un sistema operativo que se adapta mejor que Windows en máquinas de tan recursos reducidos y sus mejores prestaciones de servir páginas web.

- Software y facilidad de instalación: A parte del software lúdico, ambas plataformas cuentan con mucho software de todo tipo. Microsoft Office es un buen software ofimático, pero la alternativa de Sun llamada Staroffice un producto de mucha calidad. Aún así Windows tiene la ventaja al disponer de productos "comerciales" de más calidad que las alternativas de Linux, con las que es posible hacer un buen trabajo

Las últimas novedades de juegos no están disponibles para Linux y con los problemas por falta de una viabilidad comercial. Además, cuando estos productos están disponibles su instalación y configuración no suele ser "coser y cantar". Se establece por ejemplo las impresiones acerca de la instalación de Quake y su configuración bajo Linux, se reducía a 15 pasos con un par de re-compilaciones del núcleo de Linux de por medio. Esto ha cambiado mucho pero lo que está claro es que Windows tiene una gran ventaja en este aspecto. Lo que se quiere es instalar el juego con un doble clic y ejecutarlo con otro; eso es lo que pasa con la mayoría de personas.

- Plug and Play, detección e instalación de dispositivos: desde hace algún tiempo la instalación de dispositivos bajo Linux esta empezando a ser muy fácil, pero todavía queda mucho camino. El número de dispositivos que soporta Windows y la facilidad de instalarlos es mejor que en Linux aunque ciertamente si Linux sigue la evolución actual muy pronto habrá igualdad. Este problema casi está estancado.

- Características avanzadas vs facilidad de uso: Está claro que si se otorga a un sistema operativo de mayores posibilidades de configuración el tiempo de aprendizaje será mayor. En cuanto a posibilidades de configuración Linux brilla por su calidad, pero las capacidades de configuración a un usuario medio o público en general es difcil,

ya que más del 90% desconoce de los aspectos de configuración de este sistema operativo.

- Código abierto frente vs. Código propietario: las ventajas que presenta el código abierto son múltiples como:

- posibilidad y permiso de modificación
- distribución
- mejora y solución de errores más rápida.

Puede resultar que esto es un arma de doble filo. El caso de Windows, si en un programa existiera errores, estos programas seguirían instalables hasta que salga alguna modificación o parche de una nueva versión. En el caso de Linux los programas de código abierto, los errores se los podría detectar y corregir.

En la parte comercial, la presión es increíblemente elevada por lo que no se puede tener programas de dudosa calidad. Aún así Linux y su modelo representan un claro beneficio para usuarios medianos frente a las grandes empresas.

- Facilidad de instalación del S.O.: Este es otro apartado en el que las cosas tienden a una rápida igualdad. Cualquier distribución Mandrake Linux, que es quizá la más popular en el entorno doméstico por ser un sistema Linux muy preconfigurado y por sus herramientas de configuración tan fáciles de usar como las de Windows. Las

facilidades que también aportan los paquetes rpm ha contribuido mucho a que Linux sea cada día más fácil.

- Precio: Como todos conocen Linux es un sistema operativo de libre distribución mientras que Windows es un producto comercial. Si se quiere adquirir soporte técnico en el caso de Linux hay que pagar, mientras que ya está incluido en el precio de Windows. Al instalar Windows en más de una máquina hay que pagar licencias de uso y esto no ocurre con Linux.

- Integración: A pesar de la enorme evolución de Linux, sigue siendo un sistema en el que la integración no es un hecho, por lo menos no en la manera que se tiene de este concepto cuando se trata de Windows. Quizá esto sea una consecuencia del modelo de código abierto. Sin embargo en el mundo Linux la diversidad menor y más controlada, enriquece el sistema operativo, pero aún es necesario estandarizar más cosas para que las aplicaciones trabajen mejor una con otra. No se trata de conseguir el nivel de integración que puedan tener Microsoft Windows, Office, Explorer y Visual Studio ya que al ser todas ellas herramientas de la misma compañía esto es imposible, pero sí de avanzar en la integración.

4.8.2 Dos y Linux.

No es raro tener ambos, Linux y MS-DOS, en el mismo sistema. Muchos usuarios de Linux confían en MS-DOS para aplicaciones tales como procesadores de texto, aunque Linux proporciona sus propios análogos para estas aplicaciones (por ejemplo, TEX),

existen varias razones por las que un usuario concreto desearía correr tanto MS-DOS como Linux.

Si todo el texto esta escrito en WordPerfect para MS-DOS, puede no ser capaz de convertirla fácilmente a TEX o algún otro formato. Hay muchas aplicaciones comerciales para MS-DOS que no están disponibles para Linux, y no hay ninguna razón por la que no pueda usar ambos.

Como puede saber, MS-DOS no utiliza completamente la funcionalidad de los procesadores 80386 y 80486. Por otro lado, Linux corre completamente en el modo protegido del procesador y explota todas las características del este, puede acceder directamente a toda su memoria disponible (e incluso mas allá de la disponible, usando RAM virtual). Linux proporciona un interface Unix completo no disponible bajo MS-DOS, el desarrollo y adaptación de aplicaciones Unix bajo Linux es fácil mientras que, bajo MS-DOS, esta limitado a un pequeño subgrupo de la funcionalidad de programación Unix. Al ser Linux un sistema parecido a Unix, no tendrá estas limitaciones.

Sin embargo, basta decir que Linux y MS-DOS son entidades completamente diferentes. MS-DOS no es caro (comparado con otros sistemas operativos comerciales), y tiene un fuerte asentamiento en el mundo de los PC's. Ningún otro sistema operativo para PC ha conseguido el nivel de popularidad de MS-DOS básicamente porque el coste de otros sistemas operativos es inaccesible para la mayoría de los usuarios de PC's.

4.8.2.1 DOSEMU (Emulación de una Aplicación)

Ejecutando en línea de comando dosemu la aplicación de DOS se presenta en la siguiente manera.

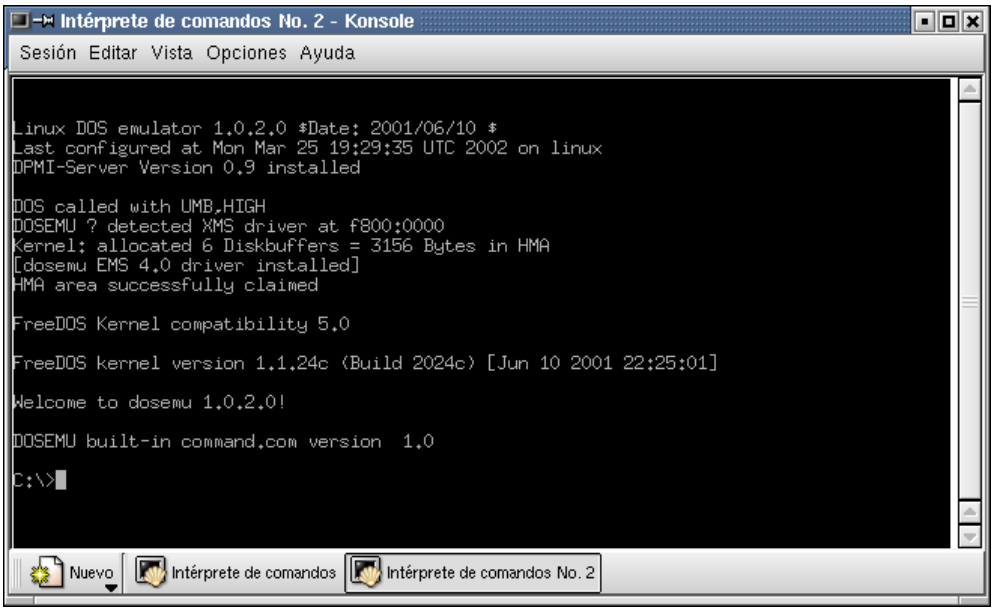


Fig # 4-15. *Pantalla inicial de DOSEMU*

Para el ejemplo siguiente ejecutamos Borland C para DOS

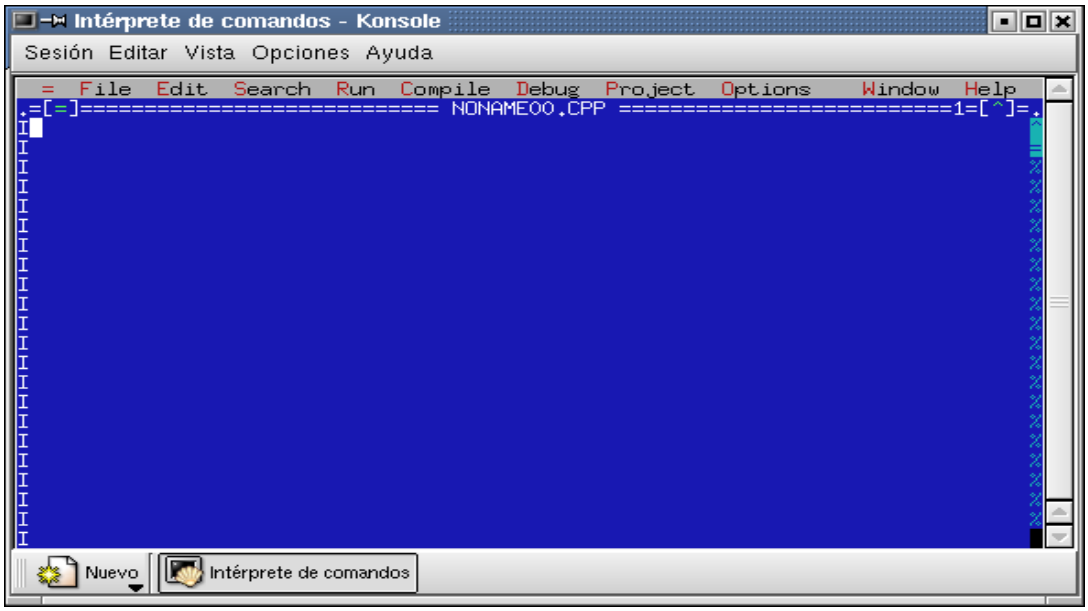


Fig # 4-16. *Ejecución de BorlandC para DOS, mediante DOSEMU*

Se dispone de herramientas que permiten interactuar entre Linux y MS-DOS. Por ejemplo, es fácil acceder a los ficheros MS-DOS desde Linux. También el emulador de MS-DOS, que permite ejecutar muchas aplicaciones populares de MS-DOS.

4.8.3 Unix - Linux

Al igual, el mantener la ejecución de una aplicación desarrollada en Unix y ejecutarla a Linux, conlleva a que Linux no es el sustituto ideal, Unix puede cumplir el papel. En tareas como comercio electrónico o finanzas de alto volumen, no hay mejor solución que una migración a una máquina multiprocesador con un Unix convencional.

El coste inicial puede parecer alto, pero los beneficios relacionados con un menor gasto en personal de administración, menos servidores para ocuparse de la carga asignada, y por supuesto su gran capacidad de proceso, harán desaparecer esas inquietudes. En muchas áreas de alta demanda, Linux no está aún a la altura de una versión de Unix de mucha demanda, pero esa distancia empieza a desaparecer rápidamente.

Los administradores de sistemas deberán aprender nuevas técnicas del sistema operativo como parte de la migración, lo cual no debería ser problemático si dichas personas son competentes. La reducción de plantilla se producirá casi con total certeza en la migración de Windows, Unix / Linux (ya que éste es uno de los beneficios centrales para una organización de la migración a Unix / Linux).

Para demostrar la compatividad de los sistemas operativos Unix / Linux, se ha procedido a compilar el código fuente de GFTP[44], en que se presenta para la plataforma solaris. Se ha escogido como sistema operativo Solaris por ser una plataforma más amigable que otras versiones de Unix.

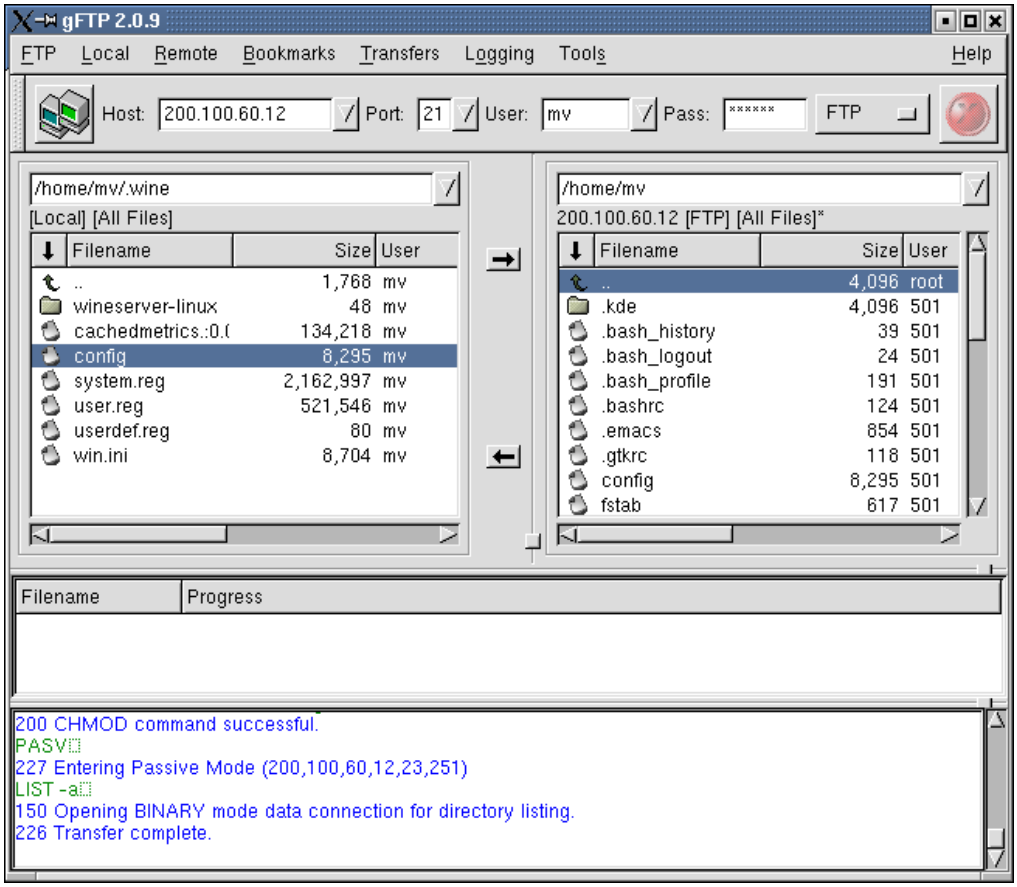


Fig # 4-17. *Ejecución de GFTP recompilado su código fuente bajo Solaris*

Para las aplicaciones que se ha propuesto demostrar se han tenido muchos inconvenientes por el sistema operativo Solares, en el que presenta mucha dependencia de librerías de actualización por los bugs del sistema operativo, librerías de enlace para

[44] GFTP: Graphics File Transfer Protocol

las aplicaciones y librerías gráficas en la que no son totalmente compatibles al compilar los códigos fuentes de aplicaciones como:

- Openoffice
- Koffice
- Abiword

Estas aplicaciones atribuyen a paquetes completamente ofimáticos en los que se ha remplazado a los presentados en el anteproyecto de tesis debido a que no están disponibles los códigos fuentes.

En las aplicaciones ofimáticas debido a su complejidad y robustez, la mucha dependencia de: librerías gráficas, librerías del sistema operativo y de las librerías propias de las aplicaciones se presentaron los siguientes inconvenientes:

- Con Koffice se pudo compilar e instalar pero la ejecución presenta un error en la aplicación.
- El abiword no compila debido a un error de código fuente, el makefile de un directorio no existe por lo que no encuentra las librerías de enlace de compilación.
- En Openoffice existe los binarios para Solaris que se instala con algunas dificultades debido a que pide muchos parches de actualización para el sistema operativo. El problema radica que en linux al momento de compilar el código fuente falta las librerías de cabecera (.h) que no existen dentro de la aplicación, al crear los archivos objeto no los puede enlazar y finaliza la recompilación sin éxito.

Para aplicaciones que no tienen mucha dependencia de librerías gráficas como GFTP, XPUYOPUYO (juego) la recompilación se realizó sin dificultad por lo que puede utilizarse en cualquier sistema operativo Unix - Linux.

Con respecto a la base de datos Postgres como no utiliza librerías gráficas se recompila sin ningún problema ya que se designa a la plataforma a la que se va a ejecutar.

4.9 Estudio de librerías precompiladas DLL.

4.9.1 Estructura de una DLL

Al tratar de la construcción de un programa se señala que en ocasiones no se desea construir un ejecutable, sino una librería que ayuden a dividir los programas en segmentos de código que contienen alguna funcionalidad pre-construida que puede ser utilizada o llamada por un ejecutable. Las librerías contienen en su interior variables y funciones encapsuladas en forma de clases.

De forma general, el término librería se utiliza para referirse a un conjunto de módulos objeto .obj (resultados de la compilación) agrupados en un solo fichero que suele tener las extensiones .LIB, .OBJ y .DLL. Estos ficheros permiten tratar las colecciones de módulos como una sola unidad, y representan una forma muy conveniente para el manejo y desarrollo de aplicaciones grandes.

Además de ser un concepto muy productivo para la industria del software, ya que permiten la existencia de las librerías de los propios compiladores y de un mercado de utilidades con componentes adicionales. Son las denominadas librerías 3pp (de terceras partes), en referencia a que no son incluidas de forma estándar con los compiladores, ni creadas por el programador de la aplicación.

En este sentido el software se parece a cualquier otro mercado de componentes, en donde las librerías más o menos extensas que acompañan a los compiladores, que permiten añadir a nuestros programas las funcionalidades más diversas sin necesidad de ser un experto en cada área de la programación y sin necesidad de estar programándolas constantemente.

4.9.1.1 Tipos

En lo que respecta al lenguaje C++, existen dos tipos fundamentales de librerías: Estáticas y Dinámicas, que aunque comparten el mismo nombre genérico "Librería" en realidad se refieren a conceptos distintos.

4.9.1.1.1 Librerías estáticas

Denominadas también librerías objeto, son colecciones de ficheros objeto .obj (compilados) agrupados en un solo fichero de extensión .LIB y .OBJ. Los prototipos de las funciones utilizadas en estas librerías, junto con algunas macros y constantes predefinidas que facilitan su uso, se agrupan en ficheros denominados "de cabecera",

porque es tradición utilizar las primeras líneas del programa para poner las directivas #include que los incluirán en el fuente durante la fase de preprocesado.

Las librerías estáticas se componen de uno o varios ficheros .lib, .obj junto con uno o varios ficheros de cabecera (generalmente .h). Durante la fase de compilación, el enlazador incluye en el ejecutable los módulos correspondientes a las funciones de librería que hayan sido utilizadas en el programa, de forma que entran a formar parte del ejecutable, de ahí su nombre; librerías enlazadas estáticamente.
El compilador Borland C++ dispone de una herramienta específica para la creación y manejo de librerías estáticas; el ejecutable TLIB.EXE.

4.9.1.1.2 Diccionario

Junto con los diversos módulos que componen, las librerías estáticas incluyen una especie de índice o diccionario con información sobre su contenido. Este diccionario sirve para incrementar la velocidad de enlazado cuando el "Linker" debe incluir alguno de sus módulos en un ejecutable, e incluye una tabla con los nombres de todos los recursos públicos (estos símbolos deben ser distintos). Por "públicos" se refiere a que pueden ser accedidos desde el exterior por los usuarios de la librería.

4.9.1.1.3 Librerías dinámicas

Otra forma de añadir funcionalidad a un ejecutable son las denominadas librerías de enlazado dinámico, generalmente conocidas como DLLs ("Dynamic Linked Library").

Estas librerías se utilizan mucho en la programación para el sistema operativo Windows en todas sus versiones.

Este sistema contiene un gran número de librerías de terminación .DLL, aunque en realidad pueden tener cualquier otra extensión como: .EXE, .FON, .BPI, .DRV etc.; de forma genérica se refiere a ellas como DLLs, nombre por el que son más conocidas.

4.9.1.1.4 Diferencias: Librería Estática versus Dinámica

Las diferencias más relevantes de las librerías dinámicas respecto a las estáticas son fundamentalmente dos:

- Las librerías estáticas quedan incluidas en el ejecutable, con lo que el tamaño de la aplicación (ejecutable) es mayor. Esto puede ser de mucha importancia en aplicaciones muy grandes, ya que el ejecutable debe ser cargado en memoria de una sola vez.

- Las librerías dinámicas son ficheros independientes que pueden ser invocados desde cualquier ejecutable, de modo que su funcionalidad puede ser compartida por varios ejecutables. Esto significa que solo se necesita una copia de cada fichero de librería (DLL) en el sistema. Esta característica constituye la razón principal de su utilización, y es también origen de algunos inconvenientes, principalmente en sistemas como Windows en los que existen centenares de ellas.

Como consecuencia de las diferencias citadas se derivan otras. Por ejemplo: si se realizan modificaciones en los módulos de una librería estática, es necesario recompilar todos los ejecutables que la utilizan, mientras esto no es necesario en el caso de una librería dinámica.

Durante la ejecución de un ejecutable, las librerías estáticas que hubiesen intervenido en su construcción no necesitan estar presentes, en cambio las dinámicas deben estar en el mismo directorio o en el camino de búsqueda ("Path").

Las librerías estáticas solo se utilizan en la fase de construcción del ejecutable. Las dinámicas se utilizan durante la ejecución.

4.9.2 Estudio de las cabeceras en las DLL´s

4.9.2.1 Construcción de una DLL

Cuando se construye un programa fuente que será compilado para producir una DLL, las funciones que deban ser accesibles desde otros ejecutables se denominan "exportables" o también "Callbacks". Esta circunstancia debe ser conocida por el compilador, por lo que en estos programas es necesario especificar que recursos se declararán como “exportables”, y además debe instruirse al "Linker" para que genere una librería dinámica en vez de un ejecutable normal.

En los compiladores para la plataforma Windows existen varias formas para declarar una función o recurso como "exportable", pero aquí se indica a dos instrucciones directas, los especificadores _export y dllexport.

En determinados casos también puede ser necesario construir una librería de importación para la librería dinámica. Esta es una librería estática clásica (.LIB) que sirve como índice o "diccionario" de la dinámica.

4.9.2.2 Tabla de entradas.

La forma utilizada en los recursos declarados como "exportables" son incluidos por el enlazador en una tabla especial, incluida en la librería, que se denomina tabla de exportación ("Export table") o tabla de entradas ("Entry table"). Esta tabla juega aquí un papel similar al diccionario de las librerías estáticas.

La tabla de exportación tiene dos tipos de datos importantes (en realidad son dos tablas):

- Los nombres con que aparecen los recursos
- Un número de orden.

Cuando una aplicación (.exe o librería dinámica) invoca una función situada en una librería, el módulo que realiza la invocación puede referirse al recurso por nombre o por

número de orden. Por lo general, la segunda forma es ligeramente más rápida, ya que no se necesitan comparaciones de cadenas para localizar la entrada.

Cuando un recurso es exportado por número, la parte de nombres de la tabla no necesita ser residente en memoria (del ejecutable que la utilizará). En cambio, si es exportada por nombres, dicha tabla si necesita ser residente, y será cargada en memoria cada vez que el módulo sea cargado.

Aunque para ahorrar memoria y mejorar el acceso, pueden utilizarse una exportación y acceso por número, Microsoft recomienda que las librerías dinámicas se exporten por nombre, de lo contrario no se garantiza que nuestras librerías sean utilizables por todas las plataformas y versiones de Windows.
Es importante recordar, sobre todo si vamos a construir DLLs que serán utilizadas por terceros, que los nombres de los recursos exportados no pueden colisionar con ningún otro nombre utilizado por el programa o por otra DLL del sistema, de forma que debemos asegurarnos que estos nombres serán únicos.

4.9.3 Llamadas a las DLL's en Linux

4.9.3.1 Utilización de librerías estáticas.

Este es el método tradicional, son las clásicas colecciones de ficheros objeto .obj o .o en el sistema operativo Linux (compilados), que en el momento de la construcción de la aplicación, son incluidos por el "Linker" en el propio ejecutable.

4.9.3.2 Utilización de librerías dinámicas.

En esta modalidad, los recursos ocupan un fichero independiente del ejecutable (fichero que puede ser utilizado por cualquier aplicación que lo necesite). En algún momento, durante la carga del ejecutable o posteriormente (en run-time), el ejecutable deberá integrar este bloque de código en su propio espacio, de forma que pueda acceder a los recursos contenidos en la librería. En los sistemas Unix / Linux establece las llamadas librerías .SO que son equivalente a las DLL de Windows.

4.9.3.3 Utilización de programas externos.

Un ejecutable puede llamar a ejecución a otro mediante varios tipos de mecanismos. El ejecutable llamado proporciona alguna funcionalidad antes de su terminación, y dispone de espacio propio de ejecución independiente del programa que lo invocó.

Es importante para comprender el funcionamiento de las DLLs o las .SO, estas librerías no son cargadas en el espacio de memoria del ejecutable, sino que tiene su propio espacio. Es decir, en el ejecutable existe un cierto "mapa" de como es está distribuida esa zona de memoria, donde están sus objetos "exportables".

Además, el hecho de que las librerías disponga de su propio espacio, tiene una importante ventaja adicional; si dos o más procesos que se están ejecutando simultáneamente en el sistema necesitan de la misma librería, esta no necesita ser

cargada dos veces en memoria, basta que ambos tengan acceso a ella y cierto conocimiento de su estructura interna[45]

En realidad lo que caracteriza a estas librerías es la forma en que es traída a ejecución. No directamente desde el shell del sistema como un ejecutable normal, sino desde otro ejecutable (puede ser incluso otra DLL o .SO), de forma parecida a como se invoca una función (una especie de función externa al programa). Por esta razón no disponen de una función main o de un módulo de inicio en el sentido clásico.

El resultado es que si desde una DLL o .SO necesitamos utilizar una funcionalidad existente en el cuerpo del programa (una función o clase), no se puede acceder a menos que dicha función sea también incluida en una DLL o .SO independiente. Naturalmente esto exige que la separación de partes del programa en tales librerías se realice después de un estudio minucioso de las funcionalidades que serán utilizadas desde cada módulo.

4.9.3.4 Utilización de recursos

La utilización de los recursos contenidos en este tipo de librerías (.DLL, .SO) requiere dos condiciones:

- Cargar en memoria el segmento de código contenido, en un espacio accesible desde el ejecutable que lo utiliza.

[45] Un perfecto ejemplo es el caso de las DLLs que contienen la API de Windows, que son cargadas una sola vez en memoria y utilizadas por casi cualquier aplicación que se corra en el Sistema

- Poder acceder al interior de este segmento de código (conocer su topografía interna) para poder utilizar su funcionalidad.

En cuanto a los puntos anteriores, existen dos formas para que el programa "cargue" la librería

- En el mismo momento de la carga del programa.
- En el momento en que se necesite alguno de sus recursos (en runtime).

En el primer caso las librerías requeridas por el ejecutable, son cargadas e inicializadas por el módulo de inicio como cualquier otro módulo del programa es decir; que serán inicializadas antes que comience la ejecución de main. Cuando la aplicación es cargada por el sistema operativo, este mira en el fichero ejecutable para ver que librerías dinámicas se necesitan, y se encarga de cargarlas también.

Este tipo de enlace se denomina enlazado estático o implícito, donde la librería dinámica es enlazada estáticamente.

En el segundo caso, la librería es cargada durante la ejecución cuando la aplicación lo necesita. Esta forma de uso se denomina de enlazado dinámico o explícito (librería dinámica enlazada dinámicamente).

Para realizar la carga, el programador dispone de algunas funciones de la API de cada sistema operativo si es necesario, que se encargan de realizar la tarea cuando él lo decide.

Cualquiera que sea la forma de carga elegida, implícita o explícita, el proceso seguido para buscar las librerías es siempre el mismo:

- En el directorio que contiene el ejecutable.
- El directorio actual de la aplicación.
- El directorio de sistema (Windows, Unix, Linux)
- El o los directorios del sistema raíz.
- Los directorios incluidos en la variable de entorno PATH del Sistema.

En caso de una carga implícita, si el sistema no encuentra el fichero (por ejemplo una .DLL) en ninguno de los sitios anteriores, se muestra un mensaje de error y la aplicación no puede ejecutarse (Fig. 19).

Fig # 4-18. *Error cuando no encuentra una librería dentro del sistema operativo.*

En caso que el fichero no se encuentre durante el proceso de carga explícita, mediante ciertas funciones a disposición del programador, entonces es potestad de este decidir si

el sistema devuelve un error. Un ejemplo tomado de una aplicación real cuando no aparece la .DLL requerida.

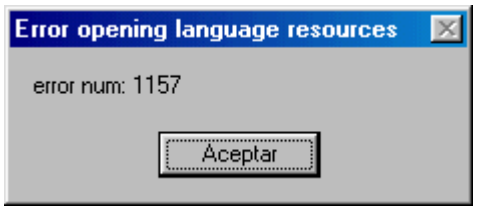


Fig # 4-19. *Error cuando no encuentra una librería dentro de la aplicación.*

Recordar que, para usar las funciones contenidas en una librería (estática o dinámica) se necesitan tres condiciones:

- Un prototipo que permita conocer el nombre del fichero que compone la librería, su localización, parámetros y tipo de retorno de la función de librería que se requiera utilizar (esto es lo normal para utilizar cualquier función).
- Disponer de los tipos de datos que pasarán como argumentos (también normal para cualquier función).
- Utilizar la convención de llamada que corresponda a la librería en cuestión. Es decir, que el enlazador C++ utilizado permita usar la convención de llamada adecuada, de forma que estos módulos externos puedan ser llamados a ejecución desde nuestro programa.

CAPITULO V

CONCLUSIONES Y RECOMENDACIONES

5.1 Verificación de la Hipótesis

La hipótesis propuesta no es totalmente verificable, existen aplicaciones que son diseñadas exclusivamente para algún sistema operativo, por lo que no pueden ser portadas hacia Linux, debería pensarse en realizar una reingeniería de software para poderla ejecutar en cualquier otra plataforma.

Aunque exista el código fuente de ciertas aplicaciones depende mucho de las API´s ya que no se podría establecer una portabilidad y escalabilidad al 100%, y dependiente de otros factores como puede ser:

- Falta de librerías de enlace
- Dependencia de la versión del sistema operativo
- El código fuente con errores de programación
- Los requerimientos mínimos que necesita la aplicación para compilar obliga a obtener nuevas versiones.
- La parte grafica (tipos de escritorios utilizados)
- La dependencia de otros paquetes informáticos.

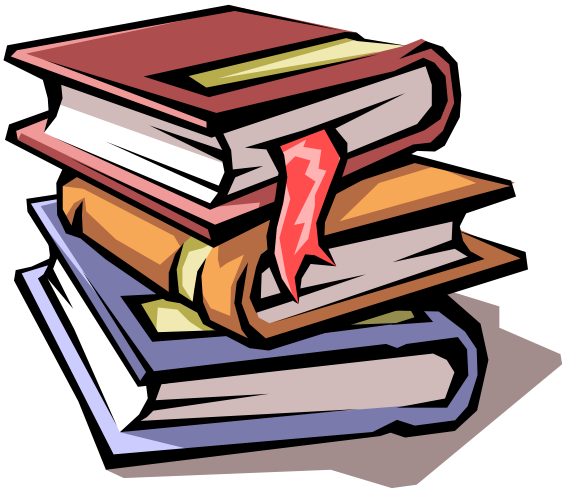
Dependiendo de la complejidad de la aplicación a ejecutar el camino más óptimo para dar una solución es utilizar un emulador o una máquina virtual de acuerdo a la necesidad; estableciendo factores como: rapidez, igualdad de manejo de la aplicación, la misma configuración, etc.

5.2 Conclusiones

- No todas las aplicaciones de Unix se ejecutan en Linux ya que, Linux no es un sustituto ideal al momento de migrar la aplicación
- A pesar de que WINE da una funcionalidad buena no ejecuta gran cantidad de aplicaciones; o se limita a unas pocas.
- La solución más optima es utilizar un software de maquina virtual por su función especifica de cargar el sistema operativo completo con las siguientes caracteristicas:
    - Interfaz de más alto nivel.
    - Extensión y simplificación de lo ofrecido por el nivel de lenguaje máquina.
    - Ocultación de gran parte de los detalles del hardware.
    - Interfaces universales para los dispositivos.
- Cada sistema operativo maneja su propia API´s con lo cual el programador escoge las herramientas necesarias en el que desea desarrollar la aplicación.
- El uso de licencias GNU para Linux conlleva a una disminución de costos.
- La disponibilidad de las fuentes permite a la mayoría de los programadores adaptar a la plataforma donde se porte, para una integración con el resto de aplicaciones de escritorio.
- La variedad de herramientas de desarrollo para las API´s y GUI de Linux permite al programador escoger el sistema operativo que desea trabajar para establecer portabilidad de la aplicación.
- La compatibilidad binaria es una solución con particularidades impredecibles, por que evidentemente siempre será preferible una versión nativa de una aplicación.
- Los tiempos de respuesta varían según la utilización de un emulador o de una máquina virtual para la aplicación a ejecutarse.

5.3 Recomendaciones

- Por el alto costo de algunas aplicaciones Windows o Unix, se recomienda ir por el software de código abierto o de licencias GNU, para obtener un igual o mejor costo beneficio. Por ejemplo, sistemas como Solaris tienen una excelente calidad, pero su costo es elevado, además necesita de características físicas muy avanzadas, lo cual limita al común de los ususarios su obtención.
- Por evitar monopolios. A los desarrolladores de software les beneficia la unificación de entornos alrededor de Windows, porque sólo tienen que hacer un único esfuerzo de desarrollo para una única plataforma. En cambio, bajo Linux es una mejor alternativa, ya que por mucha cuota de mercado que consiguiese nunca se convertirá en un monopolio, por su carácter abierto y respetuoso con los estándares, y no orientado a los intereses concretos de una sola empresa.
- Escoger el sistema operativo más adecuado para el desarrollo de sus aplicaciones sean estas de una o múltiple plataforma.
- Probar nuevas alternativas. Tal vez el mayor inconveniente sea la resistencia de nuevas ideas. A las empresas les cuesta mucho aceptar una nueva forma en que funcionan las cosas en el mundo Linux, y darse cuenta de que la mayoría de lo que a primera vista parecen inconvenientes, son en realidad ventajas.



ANEXOS

ANEXO A

Cuadro de Funciones más utilizadas en la API de Linux

Se presenta en un resumen las funciones más utilizadas en la API de Linux que son:

FD_CLR( )	asctime_r( )	cabs( )	cbrtl( )	clog( )
FD_ISSET( )	asin()	cabsf( )	ccos( )	clogf( )
FD_SET( )	asinf( )	cabsl( )	ccosf( )	clogl( )
FD_ZERO( )	asinh( )	cacos( )	ccosh( )	close( )
_Exit( )	asinhf( )	cacosf( )	ccoshf( )	closedir( )
_exit( )	asinhl( )	cacosh( )	ccoshl( )	closelog( )
_longjmp( )	asinl( )	Cacoshf( )	ccosl( )	confstr( )
_setjmp( )	assert()	Cacoshl( )	ceil()	conj( )
_tolower( )	atan()	cacosl( )	ceilf( )	conjf( )
_toupper( )	atan2()	calloc()	ceill( )	conjl( )
a64l( )	atan2f( )	carg( )	cexp( )	connect( )
abort()	atan2l( )	cargf( )	cexpf( )	copysign( )
abs()	atanf( )	cargl( )	cexpl( )	copysignf( )
accept( )	atanh( )	casin( )	cfgetispeed( )	copysignl( )
access( )	atanhf( )	casinf( )	cfgetospeed( )	cos()
acos()	atanhl( )	casinh( )	cfsetispeed( )	cosf( )
acosf( )	atanl( )	casinhf( )	cfsetospeed( )	cosh()
acosh( )	atexit( )	casinhl( )	chdir( )	coshf( )
acoshf( )	atof()	casinl( )	chmod( )	coshl( )
acoshl( )	atoi()	catan( )	chown( )	cosl( )
acosl( )	atol()	catanf( )	cimag( )	cpow( )
aio_cancel( )	atoll( )	catanh( )	cimagf( )	cpowf( )
aio_error( )	basename( )	catanhf( )	cimagl( )	cpowl( )
aio_fsync( )	bcmp( )	catanhl( )	clearerr()	cproj( )
aio_read( )	bcopy( )	catanl( )	clock( )	cprojf( )
aio_return( )	bind( )	catclose( )	clock_getcpuclockid( )	cprojl( )
aio_suspend( )	bsd_signal( )	catgets( )	clock_getres( )	creal( )
aio_write( )	bsearch()	catopen( )	clock_gettime( )	crealf( )
alarm( )	btowc( )	cbrt( )	clock_nanosleep()	creall( )
asctime()	bzero( )	cbrtf( )	clock_settime( )	creat( )
crypt( )	dirname( )	execv( )	fegetenv( )	fmin( )
csin( )	div( )	execve( )	fegetexceptflag( )	fminf( )
csinf( )	dlclose( )	execvp( )	fegetround( )	fminl( )
csinh( )	dlerror( )	exit()	feholdexcept( )	fmod()
csinhf( )	dlopen( )	exp()	feof()	fmodf( )
csinhl( )	dlsym( )	exp2( )	feraiseexcept( )	fmodl( )

csinl( )	drand48( )	exp2f( )	ferror()	fmtmsg( )
csqrt( )	dup( )	exp2l( )	fesetenv( )	fnmatch( )
csqrtf( )	dup2( )	expf( )	fesetexceptflag( )	fopen()
csqrtl( )	ecvt( )	expl( )	fesetround( )	fork( )
ctan( )	encrypt( )	expm1( )	fetestexcept( )	fpathconf( )
ctanf( )	endgrent( )	expm1f( )	feupdateenv( )	fpclassify( )
ctanh( )	endhostent( )	expm1l( )	fflush()	fprintf()
ctanhf( )	endnetent( )	fabs()	ffs( )	fputc()
ctanhl( )	endprotoent( )	fabsf( )	fgetc()	fputs()
ctanl( )	endpwent( )	fabsl( )	fgetpos( )	fputwc( )
ctermid( )	endservent( )	fattach( )	fgets()	fputws( )
ctime()	endutxent( )	fchdir( )	fgetwc( )	fread()
ctime_r( )	environ	fchmod( )	fgetws( )	free()
daylight	erand48( )	fchown( )	fileno( )	freeaddrinfo()
dbm_clearerr( )	erf( )	fclose()	flockfile( )	freopen()
dbm_close( )	erfc( )	fcntl( )	floor()	frexp()
dbm_delete( )	erfcf( )	fcvt ( )	floorf( )	frexpf( )
dbm_error( )	erfcl( )	fdatasync( )	floorl( )	frexpl( )
dbm_fetch( )	erff( )	fdetach( )	fma( )	fscanf()
dbm_firstkey( )	erfl( )	fdim( )	fmaf( )	fseek()
dbm_nextkey( )	errno	fdimf( )	fmal( )	fseeko()
dbm_open( )	execl( )	fdiml( )	fmax( )	fsetpos( )
dbm_store( )	execle( )	fdopen( )	fmaxf( )	fstat( )
difftime( )	execlp( )	feclearexcept( )	fmaxl( )	fstatvfs ( )
fsync( )	getgrgid_r( )	getpwnam_r( )	htonl( )	isblank( )
ftell()	getgrnam( )	getpwuid( )	htons( )	iscntrl()
ftello()	getgrnam_r( )	getpwuid_r( )	hypot( )	isdigit()
ftime( )	getgroups( )	getrlimit( )	hypotf( )	isfinite( )
ftok( )	gethostbyaddr( )	getrusage( )	hypotl( )	isgraph()
ftruncate( )	gethostbyname( )	gets()	iconv( )	isgreater( )
ftrylockfile( )	gethostent( )	getservbyname( )	iconv_close( )	isgreaterequal( )
ftw( )	gethostid( )	getservbyport( )	iconv_open( )	isinf( )
funlockfile( )	gethostname( )	getservent( )	if_freenameindex()	isless( )
fwide( )	getitimer( )	getsid( )	if_indextoname()	islessequal( )
fwprintf( )	getlogin( )	getsockname( )	if_nameindex()	islessgreater( )
fwrite()	getlogin_r( )	getsockopt( )	if_nametoindex	islower()

			()	
fwscanf( )	getmsg( )	getsubopt( )	ilogb( )	isnan( )
gai_strerror()	getnameinfo()	gettimeofday( )	ilogbf( )	isnormal( )
gcvt( ) l	getnetbyaddr( )	getuid( )	ilogbl( )	isprint()
getaddrinfo()	getnetbyname( )	getutxent( )	imaxabs( )	ispunct()
getc()	getnetent( )	getutxid( )	imaxdiv( )	isspace()
getc_unlocked( )	getopt( )	getutxline( )	index( )	isunordered( )
getchar()	getpeername( )	getwc( )	inet_addr( )	isupper()
getchar_unlocked( )	getpgid( )	getwchar( )	inet_ntoa( )	iswalnum( )
getcontext( )	getpgrp( )	getwd( ) l	inet_ntop()	iswalpha( )
getcwd( )	getpid( )	glob( )	inet_pton()	iswblank( )
getdate( )	getpmsg( )	globfree( )	initstate( )	iswcntrl( )
getdate_err	getppid( )	gmtime()	insque( )	iswctype( )
getegid( )	getpriority( )	gmtime_r( )	ioctl( )	iswdigit( )
getenv()	getprotobyname( )	grantpt( )	isalnum()	iswgraph( )
geteuid( )	getprotobynumber( )	h_errno	isalpha()	iswlower( )
getgid( )	getprotoent( )	hcreate( )	isascii( )	iswprint( )
getgrent( )	getpwent( )	hdestroy( )	isastream( )	iswpunct( )
getgrgid( )	getpwnam( )	hsearch( )	isatty( )	iswspace( )
iswupper( )	llroundf( )	lseek( )	mq_close( )	nexttowardf( )
iswxdigit( )	llroundl( )	lstat( )	mq_getattr( )	nexttowardl( )
isxdigit()	localeconv( )	makecontext( )	mq_notify( )	nftw( )
j0( )	localtime()	malloc()	mq_open( )	nice( )
j1( )	localtime_r( )	mblen( )	mq_receive( )	nl_langinfo( )
jn( )	lockf( )	mbrlen( )	mq_send( )	nrand48( )
jrand48( )	log()	mbrtowc( )	mq_setattr( )	ntohl( )
kill( )	log10()	mbsinit( )	mq_timedreceive( )	ntohs( )
killpg( )	log10f( )	mbsrtowcs( )	mq_timedsend( )	open( )
l64a( )	log10l( )	mbstowcs( )	mq_unlink( )	opendir( )
labs( )	log1p( )	mbtowc( )	mrand48( )	openlog( )
lchown( )	log1pf( )	memccpy( )	msgctl( )	optarg
lcong48( )	log1pl( )	memchr( )	msgget( )	opterr
ldexp()	log2( )	memcmp( )	msgrcv( )	optind
ldexpf( )	log2f( )	memcpy( )	msgsnd( )	optopt
ldexpl( )	log2l( )	memmove( )	msync( )	pathconf( )
ldiv( )	logb( )	memset( )	munlock( )	pause( )

lfind( )	logbf( )	mkdir( )	munlockall( )	pclose( )
lgamma( )	logbl( )	mkfifo( )	munmap( )	perror()
lgammaf( )	logf( )	mknod( )	nan( )	pipe( )
lgammal( )	logl( )	mkstemp( )	nanf( )	poll( )
link( )	longjmp()	mktemp( )	nanl( )	popen( )
lio_listio( )	lrand48( )	mktime()	nanosleep( )	posix_fadvise( )
listen( )	lrint( )	mlock( )	nearbyint( )	posix_fallocate( )
llabs( )	lrintf( )	mlockall( )	nearbyintf( )	posix_madvise( )
lldiv( )	lrintl( )	mmap( )	nearbyintl( )	posix_mem_offset( )
llrint( )	lround( )	modf()	nextafter( )	posix_memalign( )
llrintf( )	lroundf( )	modff( )	nextafterf( )	posix_openpt( )
llrintl( )	lroundl( )	modfl( )	nextafterl( )	posix_spawn( )
llround( )	lsearch( )	mprotect( )	nexttoward( )	posix_spawn_file_
sched_	sethostent( )	sigaltstack( )	sockatmark()	actions_addclose( )
getscheduler( )	setitimer( )	sigdelset( )	socket( )	strncmp()
sched_rr_	setjmp()	sigemptyset( )	socketpair( )	strncpy()
get_interval( )	setkey( )	sigfillset( )	sprintf()	strpbrk()
sched_	setlocale()	sighold( )	sqrt()	strptime( )
setparam( )	setlogmask( )	sigignore( )	sqrtf( )	strrchr()
sched_	setnetent( )	siginterrupt( )	sqrtl( )	strspn()
setscheduler( )	setpgid( )	sigismember( )	srand()	strstr()
sched_yield( )	setpgrp( )	siglongjmp( )	srand48( )	strtod( )
seed48( )	setpriority( )	signal( )	srandom( )	strtof( )
seekdir( )	setprotoent( )	signbit( )	sscanf()	strtoimax( )
select( )	setpwent( )	signgam	stat( )	strtok()
sem_close( )	setregid( )	sigpause( )	statvfs ( )	strtok_r( )
sem_destroy( )	setreuid( )	sigpending( )	stderr	strtol( )
sem_getvalue( )	setrlimit( )	sigprocmask( )	stdin	strtold( )
sem_init( )	setservent( )	sigqueue( )	stdout	strtoll( )
sem_open( )	setsid( )	sigrelse( )	strcasecmp( )	strtoul( )
sem_post( )	setsockopt( )	sigset( )	strcat()	strtoull( )
sem_timedwait( )	setstate( )	sigsetjmp( )	strchr()	strtoumax( )
sem_trywait( )	setuid( )	sigsuspend( )	strcmp()	strxfrm( )
sem_unlink( )	setutxent( )	sigtimedwait( )	strcoll( )	swab( )
sem_wait( )	setvbuf( )	sigwait()	strcpy()	swapcontext( )
semctl( )	shm_open( )	sigwaitinfo( )	strcspn()	swprintf( )
semget( )	shm_unlink( )	sin()	strdup( )	swscanf( )

semop( )	shmat( )	sinf( )	strerror( )	symlink( )
send( )	shmctl( )	sinh()	strerror_r( )	sync( )
sendmsg( )	shmdt( )	sinhf( )	strfmon( )	sysconf( )
sendto( )	shmget( )	sinhl( )	strftime()	syslog( )
setbuf()	shutdown( )	sinl( )	strlen()	system( )
setcontext( )	sigaction( )	sleep( )	strncasecmp( )	tan()
setegid()	sigaddset( )	snprintf( )	strncat()	tanf( )
setenv()	toupper()	vfprintf( )	wcsspn( )	tanh()
seteuid()	towctrans( )	vfscanf( )	wcsstr( )	wscanf( )
setgid( )	towlower( )	vfwprintf( )	wcstod( )	y0( )
setgrent( )	towupper( )	vfwscanf( )	wcstof( )	y1( )
tanhf( )	trunc( )	vprintf( )	wcstoimax( )	yn( )
tanhl( )	truncate( )	vscanf( )	wcstok( )	
tanl( )	truncf( )	vsnprintf( )	wcstol( )	
tcdrain( )	truncl( )	vsprintf( )	wcstold( )	
tcflow( )	tsearch( )	vsscanf( )	wcstoll( )	
tcflush( )	ttyname( )	vswprintf( )	wcstombs( )	
tcgetattr( )	ttyname_r( )	vswscanf( )	wcstoul( )	
tcgetpgrp( )	twalk( )	vwprintf( )	wcstoull( )	
tcgetsid( )	tzname	vwscanf( )	wcstoumax( )	
tcsendbreak( )	tzset( )	wait( )	wcswcs( ) l	
tcsetattr( )	ualarm( )	waitid( )	wcswidth( )	
tcsetpgrp( )	ulimit( )	waitpid( )	wcsxfrm( )	
tdelete( )	umask( )	wcrtomb( )	wctob( )	
telldir( )	uname( )	wcscat( )	wctomb( )	
tempnam( )	ungetc()	wcschr( )	wctrans( )	
tfind( )	ungetwc( )	wcscmp( )	wctype( )	
tgamma( )	unlink( )	wcscoll( )	wcwidth( )	
tgammaf( )	unlockpt( )	wcscpy( )	wmemchr( )	
tgammal( )	unsetenv()	wcscspn( )	wmemcmp( )	
time()	usleep( )	wcsftime( )	wmemcpy( )	
timer_create( )	utime( )	wcslen( )	wmemmove( )	
timer_delete( )	utimes( )	wcsncat( )	wmemset( )	
timer_getoverrun ( )	va_arg()	wcsncmp( )	wordexp( )	
timer_gettime( )		wcsncpy( )	wordfree( )	
timer_settime( )		wcspbrk( )	wprintf( )	
times( )		wcsrchr( )	write( )	
timezone		wcsrtombs( )	writev( )	

Comandos que Utilizan el API de Linux.

admin	delta	join	printf	tail	who
alias	df	kill	prs	talk	write
ar	diff	lex	ps	tee	xargs

asa	dirname	link	pwd	test	yacc
at	du	ln	qalter	time	
awk	echo	locale	qdel	touch	
basename	ed	localedef	qhold	tput	
batch	env	logger	qmove	tr	
bc	ex	logname	qmsg	true	
bg	expand	lp	qrerun	tsort	
c99	expr	ls	qrls	tty	
cal	false	m4	qselect	type	
cat	fc	mailx	qsig	ulimit	
cd	fg	make	qstat	umask	
cflow	file	man	qsub	unalias	
chgrp	find	mesg	read	uname	
chmod	fold	mkdir	renice	uncompress	
chown	fort77	mkfifo	rm	unexpand	
cksum	fuser	more	rmdel	unget	
cmp	gencat	mv	rmdir	uniq	
comm	get	newgrp	sact	unlink	
command	getconf	nice	sccs	uucp	
compress	getopts	nl	sed	uudecode	
cp	grep	nm	sh	uuencode	
crontab	hash	nohup	sleep	uustat	
csplit	head	od	sort	uux	
ctags	iconv	paste	split	val	
cut	id	patch	strings	vi	
cxref	ipcrm	pathchk	strip	wait	
date	ipcs	pax	stty	wc	
dd	jobs	pr	tabs	what	

ANEXO B

Cuadro de Funciones más utilizadas en la API de Windows

Se presenta en un resumen las funciones más utilizadas en la API de Windows (todas sus versiones Windows 9X, Windows 2000 y XP).

Nombre del Producto	Alias	Nombre DLL
CommDlgExtendedError	CommDlgExtendedError	COMDLG32.DLL
GetFileTitle	GetFileTitleA	COMDLG32.DLL
GetOpenFileName	GetOpenFileNameA	COMDLG32.DLL
GetSaveFileName	GetSaveFileNameA	COMDLG32.DLL
Arc	Arc	GDI32.DLL
BitBlt	BitBlt	GDI32.DLL
CreateBitmap	CreateBitmap	GDI32.DLL
CreateBrushIndirect	CreateBrushIndirect	GDI32.DLL
CreateCompatibleBitmap	CreateCompatibleBitmap	GDI32.DLL
CreateCompatibleDC	CreateCompatibleDC	GDI32.DLL
CreateFontIndirect	CreateFontIndirectA	GDI32.DLL
CreateHatchBrush	CreateHatchBrush	GDI32.DLL
CreatePatternBrush	CreatePatternBrush	GDI32.DLL
CreatePen	CreatePen	GDI32.DLL
CreateSolidBrush	CreateSolidBrush	GDI32.DLL
DeleteDC	DeleteDC	GDI32.DLL
DeleteObject	DeleteObject	GDI32.DLL
Ellipse	Ellipse	GDI32.DLL
EnumFontFamiliesEx	EnumFontFamiliesExA	GDI32.DLL
GetDeviceCaps	GetDeviceCaps	GDI32.DLL
GetGlyphOutline	GetGlyphOutlineA	GDI32.DLL
GetPixel	GetPixel	GDI32.DLL
GetStockObject	GetStockObject	GDI32.DLL
GetTextMetrics	GetTextMetricsA	GDI32.DLL
GetWindowDC	GetWindowDC	GDI32.DLL
LineTo	LineTo	GDI32.DLL
MoveTo	MoveToEx	GDI32.DLL
MoveToEx	MoveToEx	GDI32.DLL
PatBlt	PatBlt	GDI32.DLL
Polygon	Polygon	GDI32.DLL
Rectangle	Rectangle	GDI32.DLL
SelectObject	SelectObject	GDI32.DLL
SetBkColor	SetBkColor	GDI32.DLL
SetBkMode	SetBkMode	GDI32.DLL
SetPixel	SetPixel	GDI32.DLL

SetPixelV	SetPixelV	GDI32.DLL
SetTextAlign	SetTextAlign	GDI32.DLL
SetStretchBltMode	SetStretchBltMode	GDI32.DLL
SetTextColor	SetTextColor	GDI32.DLL
StretchBlt	StretchBlt	GDI32.DLL
TextOut	TextOutW	GDI32.DLL
CloseHandle	CloseHandle	KERNEL32.DLL
CopyMemory	RtlMoveMemory	KERNEL32.DLL
CopyMemoryToString	RtlMoveMemory	KERNEL32.DLL
CreateFile	CreateFileA	KERNEL32.DLL
DebugBreak	DebugBreak	KERNEL32.DLL
FileTimeToLocalFileTime	FileTimeToLocalFileTime	KERNEL32.DLL
FindClose	FindClose	KERNEL32.DLL
FindFirstFile	FindFirstFileA	KERNEL32.DLL
FindNextFile	FindNextFileA	KERNEL32.DLL
GetCommandLine	GetCommandLineA	KERNEL32.DLL
GetComputerName	GetComputerNameA	KERNEL32.DLL
GetCurrentThreadId	GetCurrentThreadId	KERNEL32.DLL
GetDriveType	GetDriveTypeA	KERNEL32.DLL
GetFileAttributes	GetFileAttributesA	KERNEL32.DLL
GetFileTime	GetFileTime	KERNEL32.DLL
GetProcessHeap	GetProcessHeap	KERNEL32.DLL
GetSystemDirectory	GetSystemDirectoryA	KERNEL32.DLL
GetTempPath	GetTempPathA	KERNEL32.DLL
GetTickCount	GetTickCount	KERNEL32.DLL
GetWindowsDirectory	GetWindowsDirectoryA	KERNEL32.DLL
GlobalAlloc	GlobalAlloc	KERNEL32.DLL
GlobalFree	GlobalFree	KERNEL32.DLL
GlobalLock	GlobalLock	KERNEL32.DLL
GlobalReAlloc	GlobalReAlloc	KERNEL32.DLL
GlobalSize	GlobalSize	KERNEL32.DLL
GlobalUnlock	GlobalUnlock	KERNEL32.DLL
HeapAlloc	HeapAlloc	KERNEL32.DLL
HeapFree	HeapFree	KERNEL32.DLL
HeapReAlloc	HeapReAlloc	KERNEL32.DLL
HeapSize	HeapSize	KERNEL32.DLL
LocalFileTimeToFileTime	LocalFileTimeToFileTime	KERNEL32.DLL
Lstrcat	lstrcatA	KERNEL32.DLL
lstrlenPtr	lstrlenA	KERNEL32.DLL
MulDiv	MulDiv	KERNEL32.DLL
QueryPerformanceCounter	QueryPerformanceCounter	KERNEL32.DLL
QueryPerformanceFrequency	QueryPerformanceFrequency	KERNEL32.DLL
SetFileTime	SetFileTime	KERNEL32.DLL
Sleep	Sleep	KERNEL32.DLL

CoTaskMemFree	CoTaskMemFree	OLE32.DLL
OleTranslateColor	OleTranslateColor	OLEPRO32.DLL
RegCloseKey	RegCloseKey	ADVAPI32.DLL
RegCreateKeyEx	RegCreateKeyExA	ADVAPI32.DLL
*RegOpenKeyEx	RegOpenKeyExA	ADVAPI32.DLL
RegQueryLongValue	RegQueryValueExA	ADVAPI32.DLL
RegQueryStringValue	RegQueryValueExA	ADVAPI32.DLL
RegSetLongValue	RegSetValueExA	ADVAPI32.DLL
RegSetStringValue	RegSetValueExA	ADVAPI32.DLL
SHBrowseForFolder	SHBrowseForFolder	SHELL32.DLL
ShellExecute	ShellExecuteA	SHELL32.DLL
SHGetPathFromIDList	SHGetPathFromIDList	SHELL32.DLL
AppendMenu	AppendMenuA	USER32.DLL
CallNextHookEx	CallNextHookEx	USER32.DLL
CallWindowProc	CallWindowProcA	USER32.DLL
CheckMenuItem	CheckMenuItem	USER32.DLL
CheckMenuRadioItem	CheckMenuRadioItem	USER32.DLL
ClientToScreen	ClientToScreen	USER32.DLL
CloseClipboard	CloseClipboard	USER32.DLL
CreateCaret	CreateCaret	USER32.DLL
CreateMenu	CreateMenu	USER32.DLL
CreatePopupMenu	CreatePopupMenu	USER32.DLL
DefWindowProc	DefWindowProcA	USER32.DLL
DeleteMenu	DeleteMenu	USER32.DLL
DestroyCaret	DestroyCaret	USER32.DLL
DestroyMenu	DestroyMenu	USER32.DLL
DrawEdge	DrawEdge	USER32.DLL
DrawFocusRect	DrawFocusRect	USER32.DLL
DrawFrameControl	DrawFrameControl	USER32.DLL
DrawMenuBar	DrawMenuBar	USER32.DLL
DrawState	DrawStateA	USER32.DLL
DrawText	DrawTextA	USER32.DLL
EmptyClipboard	EmptyClipboard	USER32.DLL
EnableMenuItem	EnableMenuItem	USER32.DLL
FillRect	FillRect	USER32.DLL
FindWindow	FindWindowA	USER32.DLL
FindWindowEx	FindWindowExA	USER32.DLL
GetActiveWindow	GetActiveWindow	USER32.DLL
GetAsyncKeyState	GetAsyncKeyState	USER32.DLL
GetCapture	GetCapture	USER32.DLL
GetCaretPos	GetCaretPos	USER32.DLL
GetClassName	GetClassNameA	USER32.DLL
GetClientRect	GetClientRect	USER32.DLL
GetCursorPos	GetCursorPos	USER32.DLL

GetDC	GetDC	USER32.DLL
GetDialogBaseUnits	GetDialogBaseUnits	USER32.DLL
GetDlgCtrlID	GetDlgCtrlID	USER32.DLL
GetDlgItem	GetDlgItem	USER32.DLL
GetDlgItemText	GetDlgItemTextA	USER32.DLL
GetFocus	GetFocus	USER32.DLL
GetForegroundWindow	GetForegroundWindow	USER32.DLL
GetKeyState	GetKeyState	USER32.DLL
GetMenu	GetMenu	USER32.DLL
GetMenuItemCount	GetMenuItemCount	USER32.DLL
GetMenuItemID	GetMenuItemID	USER32.DLL
GetParent	GetParent	USER32.DLL
GetProp	GetPropA	USER32.DLL
GetSubMenu	GetSubMenu	USER32.DLL
GetSystemMenu	GetSystemMenu	USER32.DLL
GetWindowLong	GetWindowLongA	USER32.DLL
GetWindowPlacement	GetWindowPlacement	USER32.DLL
GetWindowRect	GetWindowRect	USER32.DLL
GetWindowText	GetWindowTextA	USER32.DLL
GetWindowTextLength	GetWindowTextLengthA	USER32.DLL
HideCaret	HideCaret	USER32.DLL
InsertMenu	InsertMenuA	USER32.DLL
InvalidateRect	InvalidateRect	USER32.DLL
InvalidateRectPtr	InvalidateRect	USER32.DLL
InvertRect	InvertRect	USER32.DLL
LoadIcon	LoadIconA	USER32.DLL
LockWindowUpdate	LockWindowUpdate	USER32.DLL
MessageBeep	MessageBeep	USER32.DLL
OpenClipboard	OpenClipboard	USER32.DLL
PostMessage	PostMessageA	USER32.DLL
PostMessageAsAny	PostMessageA	USER32.DLL
PtInRect	PtInRect	USER32.DLL
RegisterWindowMessage	RegisterWindowMessageA	USER32.DLL
ReleaseCapture	ReleaseCapture	USER32.DLL
ReleaseDC	ReleaseDC	USER32.DLL
RemoveProp	RemovePropA	USER32.DLL
SendMessage	SendMessageA	USER32.DLL
SendMessageAsAny	SendMessageA	USER32.DLL
SendMessageAsString	SendMessageA	USER32.DLL
SetCapture	SetCapture	USER32.DLL
SetCaretPos	SetCaretPos	USER32.DLL
SetClipboardData	SetClipboardData	USER32.DLL
SetCursorPos	SetCursorPos	USER32.DLL
SetFocusAPI	SetFocus	USER32.DLL

SetMenu	SetMenu	USER32.DLL
SetMenuDefaultItem	SetMenuDefaultItem	USER32.DLL
SetParent	SetParent	USER32.DLL
SetProp	SetPropA	USER32.DLL
SetWindowLong	SetWindowLongA	USER32.DLL
SetWindowPlacement	SetWindowPlacement	USER32.DLL
SetWindowPos	SetWindowPos	USER32.DLL
SetWindowsHookEx	SetWindowsHookExA	USER32.DLL
ShowCaret	ShowCaret	USER32.DLL
ShowWindow	ShowWindow	USER32.DLL
SystemParametersInfo	SystemParametersInfoA	USER32.DLL
TrackMouseEventAPI	TrackMouseEvent	USER32.DLL
TrackPopupMenu	TrackPopupMenu	USER32.DLL
UnhookWindowsHookEx	UnhookWindowsHookEx	USER32.DLL
WindowFromPoint	WindowFromPoint	USER32.DLL

Tipos:

BITMAPFILEHEADER
BITMAPINFOHEADER
BROWSEINFO
FIXED
GLYPHMETRICS
LOGBRUSH
LOGFONT
MAT2
MINMAXINFO
NMHDR
OFNOTIFY
OPENFILENAME
PALETTEENTRY
POINTAPI
RECT
RGBQUAD
TEXTMETRIC
TRACKMOUSEEVENT
WIN32_FIND_DATA
WINDOWPLACEMENT

Constantes:

ANSI_CHARSET	BIF_DONTGOBELOWDOMAIN	CDN_FILEOK
APIFALSE	BIF_RETURNFSANCESTORS	CDN_FOLDERCHANGE
APITRUE	BIF_RETURNONLYFSDIRS	CDN_HELP
BALTIC_CHARSET	BIF_STATUSTEXT	CDN_INITDONE
BDR_RAISEDINNER	BS_DIBPATTERN	CDN_SELCHANGE
BDR_RAISEDOUTER	BS_DIBPATTERNPT	CDN_SHAREVIOLATION

BDR_SUNKENINNER	BS_HATCHED	CDN_TYPECHANGE
BDR_SUNKENOUTER	BS_HOLLOW	CREATE_ALWAYS
BF_ADJUST	BS_PATTERN	CREATE_NEW
BF_BOTTOM	BS_SOLID	DEFAULT_CHARSET
BF_BOTTOMLEFT	CBN_CLOSEUP	DEVICE_FONTTYPE
BF_BOTTOMRIGHT	CBN_SELENDOK	DRIVE_CDROM
BF_DIAGONAL	CB_ADDSTRING	DRIVE_FIXED
BF_DIAGONAL_ENDBOTTOMLEFT	CB_ERR	DRIVE_NO_ROOT_DIR
BF_DIAGONAL_ENDBOTTOMRIGHT	CB_FINDSTRING	DRIVE_RAMDISK
BF_DIAGONAL_ENDTOPLEFT	CB_FINDSTRINGEXACT	DRIVE_REMOTE
BF_DIAGONAL_ENDTOPRIGHT	CB_GETDROPPEDSTATE	DRIVE_REMOVABLE
BF_FLAT	CB_LIMITTEXT	DRIVE_UNKNOWN
BF_LEFT	CB_SELECTSTRING	DSS_DISABLED
BF_MIDDLE	CB_SETCURSEL	DSS_MONO
BF_MONO	CB_SETDROPPEDWIDTH	DSS_NORMAL
BF_RECT	CB_SETEDITSEL	DSS_RIGHT
BF_RIGHT	CB_SHOWDROPDOWN	DSS_UNION
BF_SOFT	CDM_GETFILEPATH	DST_BITMAP
BF_TOP	CDM_GETFOLDERIDLIST	DST_COMPLEX
BF_TOPLEFT	CDM_GETFOLDERPATH	DST_ICON
BF_TOPRIGHT	CDM_GETSPEC	DST_PREFIXTEXT
BIF_BROWSEFORCOMPUTER	CDM_HIDECONTROL	DST_TEXT
BIF_BROWSEFORPRINTER	CDM_SETCONTROLTEXT	DT_CENTER
BIF_BROWSEINCLUDEFILES	CDM_SETDEFEXT	DT_SINGLELINE
DT_VCENTER	FILE_SHARE_READ	HKEY_CURRENT_USER
DWL_DLGPROC	FILE_SHARE_WRITE	HKEY_DYN_DATA
DWL_MSGRESULT	GDI_ERROR	HKEY_LOCAL_MACHINE
DWL_USER	GENERIC_ALL	HKEY_PERFORMANCE_ DATA
EDGE_BUMP	GENERIC_EXECUTE	HKEY_USERS
EDGE_ETCHED	GENERIC_READ	HOVER_DEFAULT
EDGE_RAISED	GENERIC_WRITE	HS_BDIAGONAL
EDGE_SUNKEN	GGO_BEZIER	HS_CROSS
EM_CANUNDO	GGO_BITMAP	HS_DIAGCROSS
EM_EMPTYUNDOBUFFER	GGO_GRAY2_BITMAP	HS_FDIAGONAL
EM_LINEINDEX	GGO_GRAY4_BITMAP	HS_HORIZONTAL
EM_SETSEL	GGO_GRAY8_BITMAP	HS_VERTICAL
EM_SETTABSTOPS	GGO_METRICS	HTCAPTION
EM_SETTARGETDEVICE	GGO_NATIVE	HWND_BOTTOM
EM_UNDO	GREEK_CHARSET	HWND_NOTOPMOST
ERROR_SUCCESS	GWL_EXSTYLE	HWND_TOP
FILE_ATTRIBUTE_ARCHIVE	GWL_HINSTANCE	HWND_TOPMOST
FILE_ATTRIBUTE_COMPRESSED	GWL_HWNDPARENT	INVALID_HANDLE_VALUE
FILE_ATTRIBUTE_DIRECTORY	GWL_ID	KEY_ALL_ACCESS
FILE_ATTRIBUTE_ENCRYPTED	GWL_STYLE	KEY_CREATE_LINK
FILE_ATTRIBUTE_HIDDEN	GWL_USERDATA	KEY_CREATE_SUB_KEY
FILE_ATTRIBUTE_NORMAL	GWL_WNDPROC	KEY_ENUMERATE_ SUB_KEYS
FILE_ATTRIBUTE_NOT_ CONTENT_INDEXED	HC_ACTION	KEY_EXECUTE

FILE_ATTRIBUTE_OFFLINE	HC_GETNEXT	KEY_NOTIFY
FILE_ATTRIBUTE_READONLY	HC_NOREMOVE	KEY_QUERY_VALUE
FILE_ATTRIBUTE_REPARSE_POINT	HC_SKIP	KEY_READ
FILE_ATTRIBUTE_SPARSE_FILE	HC_SYSMODALOFF	KEY_SET_VALUE
FILE_ATTRIBUTE_SYSTEM	HC_SYSMODALON	KEY_WRITE
FILE_ATTRIBUTE_TEMPORARY	HKEY_CLASSES_ROOT	LB_ADDSTRING
FILE_SHARE_DELETE	HKEY_CURRENT_CONFIG	LB_ERR
LB_FINDSTRING	OFN_ENABLEINCLUDENOTIFY	RASTER_FONTTYPE
LB_FINDSTRINGEXACT	OFN_ENABLESIZING	REG_BINARY
LB_SETCURSEL	OFN_ENABLETEMPLATE	REG_DWORD
MAX_PATH	OFN_ ENABLETEMPLATEHANDLE	REG_DWORD_ BIG_ENDIAN
MF_BITMAP	OFN_EXPLORER	REG_DWORD_ LITTLE_ENDIAN
MF_BYCOMMAND	OFN_ EXTENSIONDIFFERENT	REG_EXPAND_SZ
MF_BYPOSITION	OFN_FILEMUSTEXIST	REG_FULL_RESOURCE_ DESCRIPTOR
MF_CHECKED	OFN_HIDEREADONLY	REG_LEGAL_OPTION
MF_DEFAULT	OFN_LONGNAMES	REG_LINK
MF_DISABLED	OFN_NOCHANGEDIR	REG_MULTI_SZ
MF_ENABLED	OFN_NODEREFERENCELINKS	REG_NONE
MF_GRAYED	OFN_NOLONGNAMES	REG_OPTION_BACKUP_ RESTORE
MF_HELP	OFN_NONETWORKBUTTON	REG_OPTION_ CREATE_LINK
MF_HILITE	OFN_NOREADONLYRETURN	REG_OPTION_ NON_VOLATILE
MF_MENUBARBREAK	OFN_NOTESTFILECREATE	REG_OPTION_ OPEN_LINK
MF_MENUBREAK	OFN_NOVALIDATE	REG_OPTION_ RESERVED
MF_MOUSESELECT	OFN_OVERWRITEPROMPT	REG_OPTION_ VOLATILE
MF_OWNERDRAW	OFN_PATHMUSTEXIST	REG_RESOURCE_LIST
MF_POPUP	OFN_READONLY	REG_RESOURCE_ REQUIREMENTS_LIST
MF_RIGHTJUSTIFY	OFN_SHAREAWARE	REG_SZ
MF_SEPARATOR	OFN_SHOWHELP	SC_ARRANGE
MF_STRING	OPEN_ALWAYS	SC_CLOSE
MF_SYSMENU	OPEN_EXISTING	SC_CONTEXTHELP
MF_UNCHECKED	PS_DASH	SC_DEFAULT
MF_UNHILITE	PS_DASHDOT	SC_HOTKEY
MF_USECHECKBITMAPS	PS_DASHDOTDOT	SC_HSCROLL
OEM_CHARSET	PS_DOT	SC_KEYMENU
OFN_ALLOWMULTISELECT	PS_INSIDEFRAME	SC_MAXIMIZE
OFN_CREATEPROMPT	PS_NULL	SC_MINIMIZE
OFN_ENABLEHOOK	PS_SOLID	SC_MONITORPOWER
SC_MOUSEMENU	SW_SHOW	VTA_BASELINE
SC_MOVE	SW_SHOWDEFAULT	VTA_CENTER
SC_NEXTWINDOW	SW_SHOWMAXIMIZED	WA_ACTIVE
SC_PREVWINDOW	SW_SHOWMINIMIZED	WA_CLICKACTIVE
SC_RESTORE	SW_SHOWMINNOACTIVE	WA_INACTIVE

SC_SCREENSAVE	SW_SHOWNA	WHITE_BRUSH
SC_SEPARATOR	SW_SHOWNOACTIVATE	WH_GETMESSAGE
SC_SIZE	SW_SHOWNORMAL	WH_JOURNALPLAYBACK
SC_TASKLIST	SYMBOL_CHARSET	WH_JOURNALRECORD
SC_VSCROLL	TA_BASELINE	WH_KEYBOARD
SPI_GETWORKAREA	TA_BOTTOM	WH_MOUSE
SWP_ASYNCWINDOWPOS	TA_CENTER	WH_MSGFILTER
SWP_DEFERERASE	TA_LEFT	WH_SHELL
SWP_FRAMECHANGED	TA_NOUPDATECP	WH_SYSMSGFILTER
SWP_HIDEWINDOW	TA_RIGHT	WM_ACTIVATE
SWP_NOACTIVATE	TA_RTLREADING	WM_ACTIVATEAPP
SWP_NOCOPYBITS	TA_TOP	WM_CANCELMODE
SWP_NOMOVE	TA_UPDATECP	WM_CAPTURECHANGED
SWP_NOOWNERZORDER	TME_CANCEL	WM_CHILDACTIVATE
SWP_NOREDRAW	TME_HOVER	WM_CLEAR
SWP_NOSENDCHANGING	TME_LEAVE	WM_CLOSE
SWP_NOSIZE	TME_QUERY	WM_COMMAND
SWP_NOZORDER	TRUETYPE_FONTTYPE	WM_COPY
SWP_SHOWWINDOW	TRUNCATE_EXISTING	WM_CREATE
SW_FORCEMINIMIZE	TURKISH_CHARSET	WM_CUT
SW_HIDE	UnicodeTypeLibrary	WM_DESTROY
SW_MAXIMIZE	VK_CANCEL	WM_DEVMODECHANGE
SW_MINIMIZE	VK_LBUTTON	WM_ENABLE
SW_NORMAL	VK_MBUTTON	WM_ENDSESSION
SW_RESTORE	VK_RBUTTON	WM_ERASEBKGND
WM_FONTCHANGE	WM_SETTEXT	SCR
WM_GETMINMAXINFO	WM_SETTINGCHANGE	SCRLF
WM_GETTEXT	WM_SHOWWINDOW	SCRLFCRLF
WM_GETTEXTLENGTH	WM_SIZE	SDOUBLEQUOTE
WM_HSCROLL	WM_SIZING	SEMPTY
WM_INITDIALOG	WM_SYSCOLORCHANGE	SLF
WM_INITMENU	WM_SYSCOMMAND	SLFLF
WM_INITMENUPOPUP	WM_TIMECHANGE	SPERIOD
WM_KILLFOCUS	WM_TIMER	SPIPE
WM_MENUSELECT	WM_UNDO	SSLASH
WM_MOUSEACTIVATE	WM_USER	SSPACE
WM_MOUSEHOVER	WM_VSCROLL	VBERRFILENOTFOUND
WM_MOUSELEAVE	WM_WINDOWPOSCHANGED	VBERRINVALID FILEFORMAT
WM_MOVE	WM_WINDOWPOSCHANGING	VBERRINVALID PROCEDURECALL
WM_MOVING	WPF_RESTORETOMAXIMIZED	VBERRINVALIDP ROPERTYARRAYINDEX
WM_NCCALCSIZE	WPF_SETMINPOSITION	VBERRINVALID PROPERTYVALUE
WM_NCCREATE	WS_BORDER	VBERROVERFLOW
WM_NCDESTROY	WS_CAPTION	VBERRSUB SCRIPTOUTOFRANGE
WM_NCHITTEST	WS_DLGFRAME	VBERR TYPEMISMATCH
WM_NCLBUTTONDOWN	WS_EX_APPWINDOW	

WM_NOTIFY	WS_EX_CLIENTEDGE	
WM_PAINT	WS_EX_STATICEDGE	
WM_PASTE	WS_EX_WINDOWEDGE	
WM_QUERYENDSESSION	WS_HSCROLL	
WM_QUERYOPEN	WS_OVERLAPPED	
WM_QUEUESYNC	WS_SYSMENU	
WM_QUIT	WS_THICKFRAME	
WM_SETCURSOR	WS_VSCROLL	
WM_SETFOCUS	SBACKSLASH	
WM_SETREDRAW	SCOLON	

ANEXO C

GLOSARIO

ANSI: (American National Standards Institute). Se trata del organismo estandarizador norteamericano, pero sus decisiones y normas de estandarización tienen un importante peso específico sobre la industria informática mundial. Incluye el IM (Institute of Electrical and Electronics Engineers) y la VA (Electronic Industries Association).

ASCII: American Standard Code of Information Exchange). Estándar aceptado casi mundialmente que recoge 128 caracteres, letras, números y símbolos utilizados en procesadores de textos y algunos programas de comunicaciones. Su principal ventaja es su amplia difusión y aceptación. De hecho, la mayoría de los procesadores de textos presentes en el mercado pueden importar y exportar ficheros a formato ASCII, lo que facilita el intercambio de información entre personas o empresas que no trabajan con la misma aplicación.
El más utilizado es al ASCII extendido (de 8 bits) que permite representar 256 caracteres, como la ñ, vocales acentuadas, etc., frente al ASCII de 7 bits que solo permite representar 127 caracteres.

API: Interfaz de Programación de Aplicaciones, es la interfaz por la cual una aplicación accede al sistema operativo u a otros servicios. El API es definido al nivel de código fuente y proporciona el nivel de abstracción y el Kernel (u otras utilerías privilegiadas) para asegurar la portabilidad del código.

Un API también proporciona la interfas entre el lenguaje de alto nivel y las utilerías y servicios de bajo nivel, las cuales han sido escritas sin consideración para el llamado de convenios o acuerdos soportado por los lenguajes compilados. En este caso la principal tarea del API, será la traducción de la lista de parámetros de un formato a otro y la interpretación del llamado por valor y del llamado por referencia.

BACK-UP: Copia de seguridad de los ficheros o aplicaciones disponibles en un soporte magnético (generalmente disquetes), con el fin de poder recuperar la información y las aplicaciones en caso de una avería en el disco duro, un borrado accidental o un accidente imprevisto. Es conveniente realizar copias de seguridad a intervalos temporales fijos (una vez al mes, por ejemplo), en función del trabajo y de la importancia de los datos manejados.

BASE DE DATOS: (DataBase): Conjunto de datos relacionados que se almacenan de forma que se pueda acceder a ellos de manera sencilla, con la posibilidad de relacionarlos, ordenarlos en base a diferentes criterios. Las bases de datos son uno de los grupos de aplicaciones de productividad personal más extendidos. Entre las más conocidas pueden citarse dBase, Paradox, Access y Aproach, para entornos PC, y Oracle, ADABAS, DB/2, Informix o Ingres, para sistemas medios y grandes.

BSD (Berkeley Software Distribution): Compañía de distribución de software con licencia libre.

CDE (Common Desktop Enviroment) : Primer escritorio que nación para el sistema operativo Unix alrededor de 1993, creado por IBM, Sun HP y Novell.

CGI (Common Gateway Interfase): Son ficheros escritos en cualquier lenguaje que permitan activar programas de aplicaciones desde el protocolo HTTP.

CSS (Cascade Style Sheet) : Es una especificación sobre los estilos físicos aplicables a un documento HTML y se trata de dar la separación definitiva de la lógica (estructura) y el físico (presentación) del documento.

CTSS (Compatible Time Sharing System): Un sistema operativo anterior a Unix, en el cual nace el primer sistema de tiempo compartido.

CVS (Concurrent Versions System): Sistema de elección para programadores que escriben software gratuito.

DIRECTX: Es una colección de programas que acelera el sistema en las tareas gráficas. Un componente es "Direct-3D". Con él se puede, bajo Windows, jugar de un modo más rápido y cómodo. Otros programas son, por ejemplo, "Direct-Sound" para la reproducción de sonido, así como el "Direct-Draw" y "Direct-Vídeo" para la representación de dibujos y vídeo.

DLL: (Dynamic Link Library o Share Library) Biblioteca de enlaces dinámicos. Rutinas ejecutables disponibles para aplicaciones en tiempo de ejecución. Por lo general están escritas en código reentrante de manera que puedan atender a más de una aplicación al mismo tiempo. Bajo DOS, los TSR se han usado como una manera de agregar funcionalidad en el tiempo de ejecución. Permanecen en memoria, interceptan ciertas condiciones, luego realizan su función. Los TSR nunca han sido formalmente sancionados y son susceptibles de conflicto. Sin embargo, Windows ha adoptado el método de biblioteca de enlaces dinámicos, o DLL, como una manera estándar para crear nueva funcionalidad que pueda compartirse en el sistema.

DNS : (Domain Name System). Sistema de Nombres de Dominio. El DNS un servicio de búsqueda de datos de uso general, distribuido y multiplicado. Su utilidad principal es la búsqueda de direcciones IP de sistemas centrales ("hosts") basándose en los nombres de estos. El estilo de los nombres de "hosts" utilizado actualmente en Internet es llamado "nombre de dominio". Algunos de los dominios mas importantes, que sin embargo son muy escasamente utilizados fuera de los Estados Unidos de América, son: .COM (comercial- empresas). EDU (educación, centros docentes).ORG (organización sin ánimo de lucro). NET (operación de la red). GOV (Gobierno) y MIL (ejercito). La mayoría de los países tienen un dominio propio. Por ejemplo, .US (Estados Unidos de América) .ES (España) .AU (Australia)

DOWNLOAD: Anglicismo cuyo equivalente en español es bajar. Se trata del proceso mediante el cual se carga un programa a distancia. Es de uso común en BBS, Internet y otras redes. Obtener información de la red y copiarla en nuestro ordenador.

FAT (File Access Table). Sistema de archivos utilizado en DOS y Windows

FTP : FTP son las siglas de File Transfer Protocol, el nombre del protocolo estándar de transferencia de ficheros. Su misión es permitir a los usuarios recibir y enviar ficheros de todas las máquinas que sean servidores FTP. El usuario debe disponer del software que permita hacer la transferencia (actualmente todos los navegadores, ya disponen de ese software para recibir ficheros). Los ficheros pueden ser documentos, textos, imágenes, sonidos, programas, etc., es decir, cualquier cosa que se pueda almacenar en un fichero o archivo. En Internet hay miles de ordenadores con centenares de ficheros de todas las clases a los que el público tiene acceso.

Para conectar con un servidor FTP, debemos conocer su dirección al igual que para conectar con una página Web. Hay muchos servidores FTP y por lo tanto, muchos ficheros. Para buscar estos ficheros se puede utilizar el servicio Archie. Archie reúne todos los ficheros FTP y los indexa regularmente por título y palabra clave. http://www.tile.net/tile.listserv, entre otros.

GFTP: Graphics File Transfer Protocol. Protocolo de Transferencia de Archivos en modo gráfico.

GNOME: GNU Network Object Model Enviroment.

GNU: (GNU No es Unix): Licencia Pública con la que se distribuye software.

GPL (General Public Licence): Licencia Pública General "GNU".

HARDWARE : Conjunto de componentes materiales de un sistema informático. Cada una de las partes físicas que forman un ordenador, incluidos sus periféricos. Maquinaria y equipos (CPU, discos, cintas, modem, cables, etc.). El hardware es "almacenamiento y transmisión"

HWND: Se refiere a los controladores de ventanas en Windows.

IBCS2: Intel Binary Compatibility Specification Version2

IDE: Integrate Development Enviroment: entorno de desarrollo integrado.

IEEE: Institute of Electrical and Electronics Engineers (Instituto de Ingenieros en Electricidad y Electrónica).

INTERNET: Es la red global compuesta de varias redes LAN y redes WAN que utilizan TCP/IP para proporcionar comunicaciones de ámbito mundial.

IP: Internet Protocol (Protocolo Internet)

ITU-T: International Telecommunications Union - Telecommunications (Unión Internacional de Telecomunicaciones - Telecomunicaciones).

JIT (Just in Time): En el lenguaje de programación Java, el programa JIT es el encargado de transformar los Java Bytecodes en istrucciones que pueden ser enviados y entendidos directamente al procesador.

JNI (Java Native Interface) : Es una programación nativa para Java, que permite el código escrito en JMV (java Virtual Machine) tenga interoperabilidad entre aplicaciones y librerías escritas en otro lenguaje de programación.

JPL: Laboratorio de Jets a propulsión de la NASA.

KDE (K Desktop Enviroment): Uno de los principales escritorio de Linux. Proporciona un completo entorno incluyendo gestores de archivos, ventanas, muchas herramientas y utilidades.

LAN: Red de área local.

MIDI: (Musical Instrument Digital Interface). Conjunto de estándares de 128 sonidos para tarjetas y dispositivos de sonido MIDI (sintetizadores, módulos de sonido, etc.). Mediante la asignación de instrumentos a ubicaciones de conexión MIDI específicas, el General MIDI provee un medio estándar para transmitir sonido MIDI. Debido a la pequeña cantidad de requerimiento de almacenamiento, los MIDI son muy deseables como fuente de sonido musical para aplicaciones de multimedia, en comparación con la digitalización de la música actual. Por ejemplo, un archivo MIDI de tres minutos puede ocupar sólo 20 a 30K, mientras que un archivo WAV (audio digital), podría ocupar hasta varios megabytes, dependiendo de la calidad del sonido.

MIT: Massachussets Intitute ofTechnology

MOTIF : Es una interfaz gráfica de usuario para Linux PC más ampliamente difundido por los sistemas Unix desarrollado por Metro Link Incorporated.

MWM : Motif Window Manager. Escritorio utilizado en las versiones de Linux.

NIVELES OSI :Estructura definida por ISO ,con el objeto de normalizar la estructura de las redes de computadoras. Se compone de las siguientes capas: Aplicación, Presentación, Sesión, Transporte, Red (Network), Enlace (Data Link), Físico.

OLWM: Open Look Window Manager.

OPEN LOOK: Interfaz gráfica desarollado por Sun, Xerox y ATT, en búsqueda del primer estándar de aplicaciones bajo el procesador Sparc

PLANIFICADOR: Dentro de un sistema operativo, el planificador es una parte del sistema operativo que se activa al final de cada rodaja de tiempo (gracias a los impulsos

de reloj recibidos) y decide qué proceso de los que están activos pasa a ejecutarse en la siguiente rodaja.

PLATAFORMA: Es un término de carácter genérico que designa normalmente una arquitectura de hardware, aunque también se usa a veces para sistemas operativos o para el conjunto de ambos. Los ordenadores VAX de la firma Digital, por ejemplo, serían una plataforma en la que se pueden soportar aplicaciones que, a su vez, corren (Ver: Correr) en otras plataformas.

POO: Programación Orientada a Objetos.

POSIX (Portable Operating System Interface): La Interfaz de Portabilidad de Sistemas Operativos, esvun estándar para proporcionar unos servicios de operatividad mínimos para los diversos sistemas operativos.

RED CLIENTE/SERVIDOR: (Client/Server Network). Red de comunicaciones que utiliza servidores dedicados para todos los clientes en la red. Nótese la diferencia con peer-to-peer network, que permite que cualquier cliente sea también un servidor.

RMI (Remote Method Invocation): Es una funcionalidad que Java proporciona para la construcción de aplicaciones distribuidas de la forma más transparente para el programador. Es otra implementación del concepto de RPC (Remote Procedure Call), concepto desarrollado desde hace varios años en varias plataformas.

SAMBA: Es el demonio (daemon) de Unix encargado de implementar el protocolo Netbios (SMB). Mediante este demonio se pueden compartir archivos de un sistema Unix con otras estaciones de trabajo o servidores que hablen Netbios de IBM, NetBeui (Microsoft), etc...Una de las características del Protocolo NetBios consiste en que puede transportarse de forma estándar tanto sobre TCP como sobre UDP; por lo que, en el segundo caso, podría ser muy vulnerable a ataques de suplantación de direcciones *(IP Spoofing)*.

SCO: Santa Cruz Operative System

SHELL: Conjunto de instrucciones, o macros que son utilizados en los sistemas operativos Unix – Linux.

SOCKET: Número de identificación compuesto por dos números: La dirección IP y el número de puerto TCP. En la misma red, el nº IP es el mismo, mientras el nº de puerto es el que varia

SOFTWARE: El software es "lógica y lenguaje". El software se ocupa de los detalles de un negocio en constante cambio y debe procesar transacciones en una forma lógica. Los lenguajes se utilizan para programar el software. La lógica y el lenguaje involucrados en el análisis y la programación son por lo general mucho más complejos que especificar un requerimiento de almacenamiento y de transmisión.

SVR3: System V release 3

SVR4: System V release 4

SYSTEM V: Kernel o núcleo del sistema operativo estándar desarrollado por la compañía AT&T en 1969.

TCP: Transmission Control Protocol (Protocolo de Control de Transmisión)

TIEMPO REAL: Se dice que un ordenador trabaja en tiempo real cuando realiza una transacción que le ha sido ordenada desde un terminal en ese mismo momento, sin espera alguna.

UI: User Interface (Interfaz de Usuario)

UMSDOS: Sistema de ficheros de Linux sobre una Fat

WIDGET: Tipo de componente para poder crear interfases gráficas. Estos pueden comportarse de una determinada forma dependiendo de los eventos que proporcione el usuario. Hay widgets que son ventanas, cuadros de texto, botones,etc.

WILLOWS: Empresa desarrolladora de programas que emula el API win32 de Windows.

INDICE GENERAL DE FIGURAS.

INDICE GENERAL DE TABLAS.

BIBLIOGRAFÍA

Papas Chris , Murray H (1999). *BORLAND C Y C++*. México: Mc GrawHill.

Tackett Jack Jr., Burnett Steven. (2000). *Linux 4. Edición.* Madrid: Prebtice Hall.

Ceballos Sierra Fco. Javier (1991). *Curso de Programación C ++*. Madrid: .

Petersen Richard (2001). *Fundamentos de programación en Linux*. Bogotá: McGRAW-HILL.

MANUAL DE BOLSILLO C Y C++ (1993). España: McGraw Hill .

BECERRA SANTAMARÍA CESAR A (1993). *Fundamentos de C++*. Bogotá:

DEITEL Y DEITEL (1999). *Como programar en Java*. México: PRETIENCE-HALL.

Carling M, Degler Stephen, Dennis James (1999). *ADMINISTRACIÓN DE SISTEMAS LINUX, GUIA AVANZADA.* : Prentice Hall.

Pressman Roger S. (1998). *INGENIERIA DEL SOFTWARE UN ENFOQUE PRACTICO*. MADRID: McGraw-Hill.

Business Publications España S.A. (2000 enero). Emuladores. *Pc Actual*, pp. 240.

DIRECCIONES DE INTERNET:

www.openoffice.org

www.vmware.com

www.redhat.com

www.suse.com

www.gnu.com

www.sun.com

www. linuxorg.com

www. Allapis.com

www.arrakis.es/~mapelo/

www.dosemu.org

www.kde.org

www.ciearg.com.ar

http://www.uswest.com/products/data/dsl/fast_facts.html

www.programación.net/java

http://es.tldp.org/Tutoriales/UXO/uxo/node4.html

http://www.ciberdroide.com/catalogo/P_MOTIF.html

http://www.gnulinux.org.mx/pipermail/mty-ayuda/2002-July/000517.html

http://www.mexicoextremo.com.mx/ayuda/librolinux/

http://bernia.disca.upv.es/rtportal/tutorial/01-intro/01-intro.html

www.monografías.com; Tutorial Sistemas Operativos

http://www.baquia.com/com/legacy/14390.html

http://www.iac.com.mx/computo_98/ponencia-3.html

http://es.geocities.com/ETOCANNA3/ARBOLDIRECTORIOSLINUX.htm

http://www.recursos-as400.com/wrk400/150999/api01.shtml

http://out.cgrsoftware.com/web.htm?http://www.geocities.com/practicalvb/vb/download

/win32.html

http://www.relisoft.com/win32/winnie.html

http://winapi.conclase.net/curso/index.php?000

http://www.monografias.com/trabajos7/arso/arso.shtml

http://usuarios.lycos.es/Lucho62/APIs.html

http://bochs.sourceforge.net/

http://www.tau.org.ar/base/lara.pue.udlap.mx/sistoper/capitulo8.html

http://www.tau.org.ar/base/lara.pue.udlap.mx/sistoper/capitulo11

http://www.netbsd.org/es/Goals/interop.html

http://g.unsa.edu.ar/lucas/basico/node6.html

http://www.ututo.org/kegel.html

http://www.tic.udc.es/~fbellas

Diapositivas estructuras de los sistemas Operativos, Universidad Técnica Federico Santamaría, Departamento de Informática.

http://www.readysoft.es/sic/unix-nt.htm

http://www.pue.udlap.mx/~tesis/lis/hernandez_c_ej/bibliografia.pdf

http://www.mcc.unam.mx/~cursos/Algoritmos/javaDC99-1/resumen3.html

http://lightning.prohosting.com/~rodoval/lenguajes3.html

http://www.ibiblio.org/pub/Linux/docs/linux-doc-project/install-guide/translations/es/lipp-1.1-html-2/lipp.htm

http://www.macprogramadores.org

http://www.microsoft.com/spain/servidores/windows2000/unix.asp

http://www.monografias.com/trabajos6/esin/esin.shtml

http://www.monografias.com/trabajos/sosco/sosco.shtml

www.microsoft.com

http://www.tau.org.ar/base/lara.pue.udlap.mx/sistoper/capitulo7.html

http://www.udec.cl/dti/noticias/tecnologicas/linux6.htm

http://galeon.hispavista.com/jjsoft/historia.htm

http://www.galego21.org/nos/anxo_forxan/siglas/siglas.html

http://gsyc.escet.urjc.es/docencia/asignaturas/itig-sistemas_operativos/transpas/node7.html

http://guille.costasol.net/gsaxdll/compatibilidad_ActiveX.htm

http://ve.sun.com/backissues/2002-0903/

http://www.tau.org.ar/base/lara.pue.udlap.mx/sistoper/capitulo2.html

http://216.239.37.104/search?q=cache:RvRNjh9GcqEJ:www.ibiblio.org/pub/Linux/docs/HOWTO/translations/es/ps/Umsdos-Como.ps.gz+Que+es+UMSDOS&hl=es&lr=lang_es&ie=UTF-8

http://server-die.alc.upv.es/alumno/linux/Fsstnd12/fsstnd12-5.html

http://www.yoprogramo.com/linux/linuxres1.php

http://www.willows.com/twin.html

http://216.239.53.100/search?q=cache:SVU_P_8anjEC:www.linux-magazine.com/issue/03/Wine.pdf+apis+de+windows+y+linux&hl=es&ie=UTF-8

http://www. winehq.com