UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

PROYECTO FIN DE CARRERA

ANÁLISIS E IMPLEMENTACIÓN DE UNA APLICACIÓN DE CÓMPUTO DISTRIBUIDO

Autor: David García Cano

Tutor: Julio César Hernández

Leganés, Julio de 2002

A mis padres, a mi hermana Raquel,
a los que ya no están entre nosotros
y muy especialmente a Ana.

Índice

Agradecimientos

En un momento como éste se me vienen un montón de recuerdos a la cabeza de mi paso por la universidad y por las otras instituciones en las que he estado para finalizar mis estudios, puesto que hoy, por fin, puedo ver terminada la penúltima prueba que me quedaba para finalizar la Ingeniería Técnica en Informática de Gestión.

Atrás quedan muchas horas de estudio, de prácticas, de amarguras pero, sobretodo, de felicidad y alegrías. La sensación de ir sacando adelante los estudios es una sensación muy gratificante, algo que llena a una persona de orgullo y te enseña a afrontar esta vida, la cual no se antoja nada fácil.

Es necesario decir, porque de no hacerlo no sería sincero, que para sacar estos estudios adelante siempre he tenido gente apoyándome en todo momento, tanto en los buenos como en los malos, a pesar de que en los malos se me achacaran defectos de vagueza. Pero eran estos achaques los que hacían surgir en mí una rabia interior que me ayudaba a resurgir y continuar hacia delante y superar todas las pruebas que me iba encontrando a lo largo del duro camino.

En primer lugar, me gustaría agradecer el apoyo de mis padres, y en especial el sacrificio que han hecho para costearme una educación, de lo cual siempre les estaré eternamente agradecidos y por lo que no les podré pagar jamás. También ellos, en mayor o menor medida, siempre me han apoyado en los momentos malos y me han empujado a continuar. También han celebrado los momentos más dulces a mi lado. En definitiva, han vivido todo junto a mí. Gracias, papá y mamá.

En segundo lugar, me gustaría a gradecer su apoyo, su implicación en algunos momentos de mis estudios, su ayuda en muchos trabajos, su insistencia en que estudiara más, bueno, todas esas cosas que hace una hermana mayor y que me sirvieron de mucho para poder seguir adelante. Ella siempre ha sido la persona que más ha confiado en mí, la que siempre se alegraba por mis buenas notas y la que me sermoneaba de vez en cuando con las malas, pero nunca perdió la confianza en mí y eso, a uno, le llena de satisfacción. Gracias, Raquel.

En tercer lugar, agradecer a toda esa gente de la universidad con los que he compartido horas de estudio, realizado prácticas, préstamos de apuntes, partidas de mus, conversaciones trascendentales, en definitiva, la vida social que se suele hacer en la facultad sin la cual nada habría sido lo mismo. Gracias a todos.

En cuarto lugar, agradecer a todas aquellas personas que me han ayudado a que este proyecto salga adelante, sin las cuales probablemente en la noche de hoy no estaría escribiendo este apartado. Gracias Pili por presatarme el portatil, José Manuel (Pelox), Juan, Adam,...

Y, por último, me queda una persona a la que no he podido englobar en ninguno de los grupos anteriores porque ha estado en todos; ha sido el pilar básico para que este proyecto sea lo que es, ha estado desarrollando, ha estado dando ideas, ha estado buscando información; en definitiva, ha hecho el proyecto conmigo. También me ha apoyado en mis exámenes, me ha ayudado en los estudios, se alegraba de mis éxitos, algún enfado por los fracasos, soportado mis malos humores, quedándose en casa por mis exámenes. Sin ella, nada de esto hubiera existido. Mis más sinceros agradecimientos para una persona tan especial como tú, ANA; gracias desde lo más profundo de mi corazón.

No quiero olvidarme de nadie, por lo que también agradezco su apoyo a los que ya no están, por desgracia, entre nosotros. Ellos siempre, a pesar de que alguno no me viera, me han apoyado, estén donde estén. Gracias a todos y cado uno de ellos.

Si me he olvidado de alguien, que seguro que lo he hecho, mil disculpas. Gracias a todos.

Capítulo 1

Introducción General

ÍNDICE DEL CAPÍTULO 1:

1. INTRODUCCIÓN GENERAL:

1.1 Introducción:

En vista de la acelerada marcha de la microelectrónica, la sociedad industrial se ha dado a la tarea de poner también a esa altura el desarrollo del software y los sistemas con que se manejan las computadoras. Surge la competencia internacional por el dominio del mercado de la computación, en la que se perfilan dos líderes que, sin embargo, no han podido alcanzar el nivel que se desea: la capacidad de comunicarse con la computadora en un lenguaje más cotidiano y no a través de códigos o lenguajes de control especializados.

Japón lanzó en 1983 el llamado *programa de la quinta generación de computadoras*, con los objetivos explícitos de producir máquinas con innovaciones reales en los criterios mencionados. Y en los Estados Unidos ya está en actividad un programa en desarrollo que persigue objetivos semejantes, que pueden resumirse de la siguiente manera:

- Procesamiento en paralelo mediante arquitecturas y diseños especiales y circuitos de gran velocidad.
- Manejo de lenguaje natural y sistemas de inteligencia artificial.

Es obligado comenzar esta introducción hablando de la importancia de la distribución del trabajo en diferentes equipos y de la existencia de multitud de personas que realizan proyectos de este tipo de manera desinteresada. La distribución del trabajo en diferentes ordenadores puede parecer inviable a priori.

Un pensamiento lógico que podría llegarse a tener sería el de dudar en realizar una inversión en una plataforma Cliente/Servidor con el fin de distribuir el trabajo en diferentes máquinas pudiendo invertir en una única máquina más potente que se encargará de realizar el trabajo de manera más eficiente y con menos gasto económico para la empresa. Este pensamiento puede parecer correcto, e incluso el razonamiento de esa persona puede ser lógico, pero nada más lejos de la realidad por varias razones.

En principio, diferentes comunidades informáticas están llevando a cabo diversos proyectos acerca de este tema y lo hacen desinteresadamente, sólo para ayudar a las personas a hacer descubrimientos en el mundo de la ciencia y poder conseguir logros que son impensables sin distribuir el trabajo en multitud de equipos. Al ser de manera desinteresada, le sale gratuito a la empresa, es decir, con gasto cero; el único gasto sería el ajustar sus sistemas a la plataforma que hayan seleccionado de las existentes por la red.

La distribución del trabajo en multitud de ordenadores acelera procesos que en una sola máquina tardan bastante tiempo en ser desarrollados debido a la complejidad de los cálculos que llevan asociadas las operaciones que se realizan. También existen empresas que, en lugar de invertir en equipos informáticos, hacen uso de la buena voluntad de la gente que se encuentra conectada a Internet. Estos usuarios ceden parte de la capacidad de proceso de sus CPU´s. Este espacio cedido se utiliza para realizar cálculos y ayudar a las empresas en búsqueda de soluciones de algún tipo de problema. Una vez encontrada

la solución, los propios ordenadores, sin que el usuario que está cediendo parte de su CPU se entere, informan a los servidores de la empresa de la solución y continúan con el siguiente cálculo que le manden realizar. Lo único que tienen que hacer las personas que quieran colaborar en este tipo de proyecto es instalarse el programa cliente en sus ordenadores y lanzarlo cuando no hagan uso de sus equipos, o bien cuando el uso no sea muy elevado.

Proyectos del calibre del descifrado del significado del ADN humano han sido realizados de esta manera. Personas de todo el mundo, desinteresadamente, se instalaban en los ordenadores de sus casas la aplicación Cliente y, cuando no hacían demasiado uso de su ordenador, lanzaban este programa que ayudaba a encontrar la solución del ADN humano.

Las personas que no estén familiarizadas con el tema pueden no entender para qué se toma la gente tantas molestias en distribuir el trabajo; bastaría con adquirir un ordenador más potente para realizar los cálculos y, de este modo, evitar conexiones por la red y demás inconvenientes que implica el distribuir un proceso.

Con un ejemplo claro trataremos de cambiar el parecer de cualquier persona que pueda leer este proyecto:

Imaginemos que tenemos un simple Pentium II a 200 Mghz en casa y que una empresa se plantea la posibilidad de hacer una pequeña inversión en este tipo de plataformas para distribuir el trabajo en lugar de comprar una máquina con una capacidad de proceso altísima. Ahora bien, pensemos que nuestro mismo ordenador se encuentra en miles de hogares de todo el mundo. En esos hogares, seguro que hay personas que quieren intervenir, de manera desinteresada, en proyectos para el avance de la humanidad. Lo único que tienen que hacer es ejecutar un simple programa en sus ordenadores. Pues bien, si multiplicamos la capacidad de proceso de un Pentium II por todos esos hogares en todo el mundo que están dispuestos a colaborar, nos encontramos con una capacidad de proceso que ninguna máquina en la actualidad puede llegar a conseguir.

Todos los proyectos que hacen uso de estos sistemas distribuidos son proyectos que requieren realizar cálculos matemáticos muy grandes y que máquinas tan pequeñas son imposibles de abordar. Léase, por ejemplo, en materia de seguridad informática, los cálculos tan impresionantes que hay que realizar para conseguir romper un sistema de cifrado, o cosas tan irreales como la búsqueda de indicio de vida en otros planetas.

Resulta algo extraño pensar que existan patrones que nos pueden indicar la existencia de vida en otros planetas, pero es cierto. Matemáticamente, si se dan una serie de condiciones específicas, se puede considerar la existencia de vida en otros lugares del universo. Pero, para estos cálculos, se requiere un potencial de cálculo grandísimo; de ahí estas posibilidades de distribuir el trabajo.

Por estas razones, y por muchas otras que no vienen al caso, se ha llevado a cabo este proyecto. Si todos ponemos nuestro granito de arena, conseguiremos avances importantes en el mundo de la ciencia, como la medicina, la astrofísica, la física, la informática, etc.

1.2 Marco del Proyecto:

Como ya se ha comentado en la introducción, parece casi obligatorio distribuir el trabajo en diferentes equipos informáticos para poder llevar a cabo proyectos que requieren grandes capacidades de cálculo y cómputo.

Inicialmente la idea de este proyecto surgió como algo lejano; sin embargo, debido a la importancia que parecía tener y la relevancia de las cosas que se podían llegar a obtener, sobretodo en el mundo de la seguridad informática, se comenzó a trabajar en la idea que, con el paso del tiempo, ha ido cobrando forma.

Después de muchas búsquedas por Internet, se dio por fin con una plataforma que se ajustaba bastante a la idea que se tenía acerca del proyecto que se pretendía desarrollar en la Universidad Carlos III.

De esta plataforma se hablará más adelante en posteriores capítulos; tan sólo avanzar que se trata de una plataforma diseñada para que la gente la utilice de manera libre y que ya ha dado sus frutos en terrenos como las materias químicas y otros estudios.

Con todo esto, la intención del proyecto es tener una plataforma Cliente/Servidor que funcione a pleno rendimiento en los laboratorios de la universidad, e integrarle una serie de aplicaciones informáticas desarrolladas por otras personas para poder distribuir el trabajo y conseguir resultados más rápidamente, así como llegar hasta puntos inalcanzables trabajando en un solo equipo informático a la vez.

Este proyecto está enteramente desarrollado en lenguaje C y Visual C++. El por qué de este lenguaje no es otro que el conseguir que la plataforma funcione para diversos sistemas operativos, en nuestro caso Windows NT y LINUX.

Siendo concisos, la plataforma Cliente/Servidor ha sido desarrollada enteramente en C en un entorno UNIX. Para ser compilada, en LINUX no dio ningún problema, pero para la versión de Windows las cosas no fueron tan sencillas.

Gracias a Adam L. Beberg, persona sin la que este proyecto no hubiera sido capaz de finalizar y gran partícipe en la creación de la plataforma utilizada, conseguimos sacar la versión para Windows tras una serie de intentos.

Al existir en el código comandos específicos del sistema UNIX, se tuvo que hacer uso de las herramientas cygwin, las cuales no dieron los resultados esperados. Por eso, se buscaron otras alternativas.

Estas otras alternativas pasaban por actualizar la versión de MSVC empleada que, por aquella época, era el DEV-C++. Se optó por este MSVC al tratarse de una versión gratuita disponible en Internet, pero era incapaz de interpretar comandos propios del sistema UNIX.

Por esta razón, nos decantamos por adquirir alguna versión más moderna y que fuera gratuita, pero tampoco hubo suerte.

Todas las versiones existentes eran productos no gratuitos. Finalmente, se consiguió una versión gratuita del VC++ 6.0.

Una vez conseguida esta versión, se pudo sacar la versión para Windows NT que se necesitaba.

1.3 Objetivo:

El objetivo de este proyecto no es otro que el de poner a disposición de la Universidad Carlos III una plataforma Cliente / Servidor que sirva para diversas aplicaciones y poder así distribuir la carga de trabajo, de tal modo que se puedan hacer estudios e investigaciones en diversas áreas que requieren una gran capacidad de cómputo y cálculo.

Una vez que se ha obtenido esta plataforma, lo único que se debe hacer es integrarla con la diversidad de aplicaciones que se pretendan usar para las numerosas investigaciones existentes en los diferentes laboratorios.

Si pensamos en la seguridad informática, área que hace gala y uso de cálculos con números muy grandes y específicos, llegaremos a la conclusión de que tener una plataforma de este calibre se hace casi esencial para avanzar en los estudios que se están llevando a cabo.

Por todo esto y otros factores más, es necesario que exista una gran documentación de esta plataforma para que, en un futuro y si la gente así lo desea, pueda hacer uso de ella con plena confianza y con todas las prestaciones y no se pierda en un código tan enrevesado como pueda llegar a ser el desarrollado en C.

Para probar esta plataforma, también se han desarrollado una serie de pequeños programas que intentan resolver problemas matemáticos existentes en la actualidad para los que aún no se ha encontrado explicación y que, con una distribución del cálculo en diferentes equipos, se puede avanzar más rápida y lejanamente y conseguir conjeturar algo más de lo ya existente.

Sobre estos problemas matemáticos hablaremos en próximos capítulos de este proyecto.

1.3.1 Plataforma Hardware:

En primer lugar, es necesario un servidor bastante potente que sea capaz de distribuir el trabajo entre los diferentes clientes conectados a él de manera eficiente.

Como cabe de esperar, también son necesarios diversos equipos en los que se instale el programa Cliente para poder llevar a cabo los cálculos que el servidor le ordene. Esta serie de equipos no tienen que tener unas características especiales. Está claro que, cuanto mejor sean los equipos en los que se instale el cliente, más rápido y mejor se avanzará en la solución, pero no es algo crítico para el desarrollo del trabajo.

El objetivo de este proyecto es, por tanto, realizar una plataforma Cliente/Servidor para la distribución de la carga de trabajo lo más eficiente y robusta que sea posible, además de ser portable para cualquier sistema operativo.

Para ello, el lenguaje C nos proporciona esa portabilidad y eficiencia que estamos buscando.

1.3.2 Implementación del Sistema:

A lo largo de todo el desarrollo, se pretende alcanzar la modularización. Todo proyecto Software debe tener un código legible, fácil de entender, documentado y estándar en todas las partes del programa. Éste ha sido un objetivo perseguido en toda la codificación.

El código está modularizado por librerías. Cada una de estas librerías se ocupa de una parte específica dentro de toda la plataforma Cliente/Servidor desarrollada. Con ello se obtiene la ventaja por la cual el funcionamiento de determinadas acciones es siempre el mismo y, en caso de producirse un error, estará mucho más controlado repercutiendo en un ámbito mucho más reducido.

La misma metodología llevada a cabo para la implementación de la arquitectura Cliente/Servidor ha sido utilizada para el desarrollo de los pequeños programas que tratan de resolver una serie de problemas matemáticos y de los que se ha hecho uso para testar la plataforma utilizada.

Con todas esta premisas, parece que el entendimiento del sistema por parte de terceras personas que no han intervenido para nada en este proyecto está logrado. Éste era uno de los grandes objetivos marcados al inicio del proyecto, puesto que se pretende hacer uso de la plataforma para proyectos futuros dentro de la universidad.

A parte de que terceras personas pretendan entender lo desarrollado por otros, también se logra, de esta forma, que las personas que han intervenido en este proyecto no pierdan el hilo del mismo con el transcurso del tiempo, objetivo importantísimo para posteriores mejoras y actualizaciones.

1.4 Estructura del Proyecto:

Este proyecto se desarrollará por el método de ciclo de vida en espiral. Prácticamente todos los proyectos informáticos de este tipo tienen este tipo de ciclo de vida.

Se van limando asperezas en el proyecto a medida que avanza en el tiempo y el usuario va definiendo lo que quiere en cada momento.

Las distintas etapas del proyecto serán afrontadas de manera secuencial, siendo cada una de estas etapas sometida a pruebas las cuales, previamente, habrán sido pensadas y diseñadas conscientemente para poder abarcar todos los posibles errores y situaciones existentes en el marco del presente trabajo.

La estructura de este proyecto pretende reflejar el ciclo de vida, correspondiendo cada capítulo con una etapa del desarrollo del mismo.
Con esta estructura se desea destacar la importancia de la metodología de trabajo seguida.

A continuación, se describe someramente el contenido de cada uno de los capítulos que componen esta memoria:

- **Capítulo 1.**
  El primer capítulo presenta una breve introducción del trabajo realizado, aportando motivos causantes del desarrollo del mismo, objetivos planteados y visión global del desarrollo del proyecto. Se describen brevemente la metodología de trabajo seguida y el soporte Software y Hardware utilizados para la consecución de todos los objetivos marcados.

- **Capítulo 2.**
  En este capítulo, se presentan y describen los aspectos intrínsecos de la problemática del proyecto. Se hace una descripción bastante concisa y extendida del modelo Cliente/Servidor, incluyendo un ejemplo de implementación real de una plataforma para distribuir ficheros, como es el caso de Gnutella.

- **Capítulo 3.**
  El tercer capítulo muestra la plataforma Cliente/Servidor llevada a cabo en el presente proyecto. Se habla de aspectos tan dispares como la evolución de la misma, características seguidas para el nombre de las variables utilizadas en su codificación, la presentación del código de las diversas funciones, etc.

- **Capítulo 4.**
  En este capítulo se enumeran las pruebas y los resultados obtenidos a través de las mismas para mostrar los beneficios que da el tener una plataforma Cliente/Servidor como la que se describe en este proyecto.

🗁 **Capítulo 5.**

En el capítulo 5 y último de este trabajo se exponen las conclusiones obtenidas como consecuencia de la realización del proyecto. Las conclusiones serán tanto específicas de la plataforma como generales acerca del sistema global realizado y su posible repercusión en el entorno. Para terminar, se propondrán futuras líneas de trabajo indicando los aspectos mejorables o que se pueden desarrollar en mayor medida y que mejorarían los resultados obtenidos en este proyecto.

1.5 Metodología de Trabajo:

En este proyecto, dependiendo de la fase del mismo, de los objetivos a conseguir, etc., se han utilizado una serie de equipos informáticos, más o menos potentes, en los cuales se han instalado todas las herramientas necesarias para su desarrollo.

1.5.1 Herramientas Software:

Muchos han sido los programas utilizados para el desarrollo del presente trabajo.

Se necesita tener instalados dos tipos de sistemas operativos como son Windows NT y la versión LINUX de DEBIAN.

El principal programa utilizado ha sido el MSVC Visual C++ 6.0 versión Enterprise. Este compilador de C nos permite utilizar comandos propios del sistema operativo UNIX en el sistema operativo Windows NT sin necesidad de hacer uso de ninguna herramienta que nos permita portar esta serie de comandos.

Para el desarrollo de la aplicación, no sólo se ha hecho uso de Visual C++ 6.0 Enterprise Edition, sino que también se ha hecho uso del editor de textos XEMACS que trae por defecto el sistema operativo UNIX junto con su compilador de C (comando cc).

Otros programas, de menor importancia, que se han utilizado han sido los siguientes:

- **CuteFtp versión 3.0**: programa que nos permite hacer, de manera fácil y sencilla, FTP en entornos Windows para transferir ficheros de unos equipos informáticos a otros.
- **Windows GREP versión 2.0**: sencillamente es la herramienta Grep de UNIX (como el Find de Windows) portada a Windows. Tiene un amigable interfaz gráfico y permite buscar patrones de cadenas dentro los ficheros que se deseen. Mira dentro de cualquier tipo de fichero, incluso de archivos comprimidos, por ejemplo, con formato ZIP.
- **Internet Explorer / Netscape 4.0**: programas navegadores o browser de Internet de los que se ha hecho uso para diversas funciones, entre la que cabe destacar la búsqueda de información referente al tema que nos ocupa en el presente proyecto.
- **Paquete OFFICE 97**: herramientas ofimáticas que han servido para documentar el presente proyecto. Por motivos de seguridad y confianza, este paquete ofimático ha sido instalado en un equipo informático con un sistema operativo Windows 98.

A parte de las herramientas anteriormente comentadas, se ha hecho uso de otras, cuya importancia y relevancia es mínima para el presente proyecto, por lo cual no tienen cabida en el punto que estamos desarrollando.

1.5.2 Soporte Hardware:

Como ya se ha comentado, el proyecto ha sido llevado a cabo bajo dos sistemas operativos, Windows NT y LINUX de DEBIAN.

La razón de utilizar ambos sistemas operativos no es otra que obtener la plataforma Cliente/Servidor apta para ambos sistemas operativos que, en la actualidad, son los más demandados en el mercado y en la comunidad informática. A pesar de haber utilizado estos dos sistemas operativos, la plataforma está desarrollada para poder ser usada en una gran diversidad de sistemas operativos como por ejemplo Solaris, MacOS, etc.

Si el proyecto se quiere implantar a gran escala dentro de los laboratorios de la propia universidad, se necesitaría tener un servidor potente, capaz de distribuir el trabajo entre los diferentes clientes.

El sistema operativo sobre el cual se tendría que operar sería uno totalmente adaptado a la red existente, con una tasa de fallos mínima.

Unix y Windows NT son las dos opciones que existen con más fuerza en el mercado actual y con capacidad contrastada. En diferentes empresas, la máquina principal está montada sobre Unix. Otras veces, las empresas hacen uso de Windows NT y, en raras ocasiones y sólo cuando no existan muchos clientes que remitan peticiones al servidor, se utiliza Windows 95 y Windows 98.

Por otro lado, debe de existir una red potente pero sin excesivos costes. Debe de disfrutar de un ancho de banda adecuado para no ralentizar en exceso las peticiones de descarga de trabajo que realiza el servidor al cliente y poder obtener así resultados más rápidamente.

Previamente, las pruebas fueron realizadas con el interfaz de loopback, es decir, la IP 127.0.0.1. Es una dirección fija que debe de existir en toda máquina y representa el loopback, el cual es un interfaz virtual con el que la máquina se conecta consigo misma, por ejemplo, para comprobar la configuración. De este modo, no es necesario ningún tipo de instalación fuera de lo normal. Dependiendo su uso futuro, habrá que adecuar las instalaciones para la plataforma haciendo uso de diversos mecanismos de comunicación como puedan ser Hubs, Switch, etc.

Por último, los clientes pueden estar instalados tanto en Windows NT como en Unix, pero sería recomendable que el sistema operativo utilizado para el servidor sea el mismo para el de los clientes, aunque la plataforma Cliente/Servidor llevada a cabo tiene un desarrollo tan preciso que no se vería afectada por la diferencia de sistemas operativos existentes entre los clientes y el servidor.

1.5.3 Desarrollo del Proyecto:

El desarrollo de este proyecto ha sido en un entorno de trabajo que no es el que finalmente se utilizará cuando sufra una explotación real.
Sin embargo, sí son comprobables y totalmente fiables los resultados obtenidos en el presente proyecto.

Si la plataforma Cliente/Servidor funciona en el entorno en el cual se ha desarrollado, debe funcionar de igual modo bajo otro diseño de arquitectura. La filosofía es la misma: un servidor, muchos clientes, una red que les conecta, etc.
Por tanto, se puede asegurar que la plataforma es robusta y portable.

No obstante, para una correcta instalación y funcionamiento de la plataforma Cliente/Servidor se deben parametrizar ciertos aspectos.

Ha de realizarse un *tuning* o ajuste de la plataforma, esto es, optimizar el rendimiento, consultas y tiempo de respuesta tanto del servidor como de los diferentes clientes utilizados.
La mejor manera de realizar este proceso consiste en la experiencia. Basándonos en el funcionamiento actual del sistema, se pueden detectar ciertos letargos o puntos críticos de mejora.

En cualquier caso, la implantación de la plataforma Cliente/Servidor es un punto clave en el funcionamiento del mismo. A través ésta, la distribución de la carga de trabajo debe ser una alternativa a los métodos existentes en la actualidad.

Capítulo 2

Estado de la Cuestión

ÍNDICE DEL CAPÍTULO 2:

2. ESTADO DE LA CUESTIÓN:

2.1 Introducción:

En este capítulo se pretende explicar todo lo referente a las plataformas Cliente/Servidor, tanto su contenido teórico como algunos ejemplos existentes en la actualidad.

El completo entendimiento de este tipo de arquitectura y su correcta elección para el caso que nos ocupa, es decir, la distribución de carga de trabajo en diferentes equipos apoyados en un servidor, hay que tenerla estudiada de antemano. Una buena elección ha de permitir dejar abierta la posibilidad de futuras ampliaciones y permitir que el sistema sea portable a otro tipo de plataforma, entendiendo como plataforma al Sistema Operativo en el que se base la arquitectura formada. Una plataforma Cliente/Servidor óptima será aquella que sirva para diversos sistemas operativos, como por ejemplo Win32, WinNT, Unix, Linux, Solaris, etc.

Todos estos aspectos serán desarrollados a lo largo de este capítulo con más precisión y se intentará dejar claro al lector el problema que nos ocupa.

2.2 Tecnología:

2.2.1 Introducción:

Dentro de cualquier organización en la actualidad, los sistemas de información se apoyan en una infraestructura informática. Esta infraestructura ha estado ligada en el pasado al propio modelo de la organización.

Tradicionalmente, las organizaciones han tenido una estructura centralizada y jerárquica, dividida en unos departamentos con cometidos concretos. Las relaciones entre los distintos departamentos, dentro de la jerarquía, estaban perfectamente definidas.

El modelo actual de organización, por el contrario, se articula según unas unidades más operativas y autónomas que funcionan por cumplimientos de objetivos. Existen menos niveles jerárquicos y las relaciones que existen entre las distintas unidades son más directas y pueden variar con el tiempo. Pero, por otra parte, se tiende a centralizar los datos corporativos que son importantes desde el punto de vista estratégico.

Debido a esto, la infraestructura informática se ha dividido, históricamente, en dos tipos de arquitecturas claramente diferenciadas:

- Arquitectura Centralizada: existe un servidor central donde residen todos los datos y tratamientos de los mismos.
- Arquitectura Distribuida: la inteligencia está distribuida en diferentes máquinas y los datos pueden estar centralizados en diferentes ordenadores.

2.2.2 Arquitecturas:

2.2.2.1 Arquitectura Centralizada:

Nace en torno a una concepción tradicional de la organización, con estructura centralizada y jerárquica dividida en departamentos. Cada departamento tiene unas actividades muy claras y muy concretas. Las relaciones que se puedan establecer entre los diferentes departamentos están muy definidas y limitadas y suelen realizarse a través de la jerarquía.

El sistema informático es único y la arquitectura está centralizada en un servidor central al que sólo tienen acceso los usuarios del departamento correspondiente.

- Características funcionales:
  - El ordenador central es el único de la organización.
  - Él contiene todos los datos y es el responsable de la consolidación de la información.
  - Desde el ordenador central se controla el acceso a múltiples terminales conectados a través de productos integrados en la arquitectura de red del suministrador.
  - Los terminales funcionan como “esclavos” del ordenador central.
  - Cada usuario tiene un número asignado con derechos y prioridades de ejecución en la máquina de sus programas o peticiones.

❖ Características físicas:
- Único ordenador corporativo dimensionado para soportar todos los procesos de la organización, todos los datos y las posibles comunicaciones con las delegaciones.
- Una gran base de datos donde residen todos los datos del organismo.
- Impresoras y terminales (u ordenadores personales con emulación de terminal) como puestos de trabajo conectados en grupos (clusters) al ordenador central.

❖ Características lógicas:
- Ejecución de todos los procesos en el ordenador corporativo.
- Si la empresa está dispersa geográficamente y dispone de comunicaciones, todos los puestos de trabajo están conectados al ordenador formando una estrella.

❖ Principales ventajas:
- Alto rendimiento transaccional.
- Alta disponibilidad.
- Entorno probado y personal experimentado.
- Control total del ordenador, al ser éste único y residente en un solo Centro de Proceso de Datos.
- Concentración de todo el personal de explotación y administración del sistema en un único Centro de Proceso de Datos.

❖ Principales Inconvenientes:
- Alto precio del ordenador al requerirse mucha potencia de tratamiento para dar servicio a todos los usuarios que estén conectados y gran espacio de disco para albergar todos los datos del organismo.
- Alta dependencia de las comunicaciones existentes. En caso de caída de una línea, todos los puestos de trabajo dependientes de dicha línea quedan inoperantes.
- Interfaces de usuario de caracteres (no gráficos) y, por tanto, poco amigables.
- Arquitecturas propietarias.

2.2.2.2 Arquitectura Distribuida:

Surge con los nuevos modelos organizativos en los que la empresa se divide en unidades, más o menos autónomas, que establecen relaciones más definidas y directas entre sí.

Aparecen entonces entornos informáticos departamentales adecuados a las necesidades de cada departamento en concreto.

Un sistema distribuido es un caso especial de una red de computadoras. Interconecta los lugares que tienen recursos computacionales para capturar y almacenar datos, procesarlos y enviar datos e información a otros sistemas, tales como un sistema central.

El rango de recursos computacionales varía. Algunos lugares utilizan terminales, otros microcomputadoras, otros incluso grandes sistemas de cómputo. No existe el requisito de que todo el equipo tiene que ser del mismo fabricante. De hecho, se espera que estén

implicadas varias marcas hardware. Esto permite al usuario tener el tipo más adecuado a sus necesidades.

Todos los lugares (reciben el nombre de nodos en el procesamiento distribuido) tienen la capacidad de capturar y procesar datos donde ocurran eventos. En otras palabras, si un lugar específico usa una microcomputadora, los usuarios capturan y procesan datos en su microcomputadora. Reciben respuestas rápidas a sus consultas, almacenan datos en el sistema y preparan reportes cuando se necesiten. Sin embargo, también pueden transmitir datos o reportes desde su sistema a otro enlazado en la red compuesta por todos los sistemas interconectados.

Un sistema de procesamiento distribuido incluye:

- Múltiples componentes de procesamiento de propósito general. Pueden asignarse tareas específicas a los sistemas de procesamiento sobre una base dinámica. Los sistemas no necesitan ser de una misma marca o tamaño.
- Sistema operativo de alto nivel. Los nodos de procesamiento individual tienen su propio sistema operativo, el cual está diseñado para la computadora específica. Pero también hay sistemas operativos que los enlaza e integra al control de los componentes distribuidos.
- Distribución física de componentes. Las computadoras y otras unidades de procesamiento están separadas físicamente. Interactúan entre sí por medio de una red de comunicaciones.
- Transparencia del sistema. Los usuarios no conocen la ubicación de un componente en el sistema distribuido o nada de su fabricante, modelo, sistema operativo local, velocidad o tamaño. Todos los servicios se piden por su nombre.
- El sistema operativo distribuido lleva a cabo todas las actividades que implican la ubicación física y atributos de procesamiento para satisfacer la demanda del usuario.
- Papel dual de los componentes. Los componentes individuales de procesamiento pueden operar independientemente del marco de trabajo del usuario.
- Están fuera de la clasificación como sistemas distribuidos los siguientes:
  - Una computadora multifuncional grande que distribuye el procesamiento entre diferentes procesadores de entrada/salida y periféricos.
  - Un procesador primario que controla las comunicaciones del sistema al cual fue añadido.
  - Un conjunto de terminales remotas que recogen y transmiten datos a un sistema anfitrión.
  - La interconexión de varias computadoras anfitrionas que transmiten mensajes y llevan a cabo funciones y tareas exclusivas.
  - Una computadora que puede ser particionada, es decir, que es capaz de operar en diversas sesiones de procesamiento de forma simultanea utilizando un sistema operativo especial.
- La diferencia de una red de computadoras y un sistema distribuido es que en una red de ordenadores el usuario se conecta explícitamente en otra máquina, lanza tareas remotas, mueve archivos, etc. La diferencia radica en quién invoca las funciones del sistema.

- Síntomas de Distribución:
  - Multiproceso (concurrencia): el hardware permite el progreso simultaneo de varias actividades (varias CPU´s con memoria local, etc.).
  - Interconexión: permite la comunicación entre las actividades.
  - Relación: uso compartido de recursos, de información, etc.
  - Fallo independiente: permite buscar soluciones resistentes en caso de fallo (hay que tener en cuenta que las comunicaciones también pueden fallar).

- Propiedades:
  - Nombrado global: el mismo nombre es válido en todo el sistema.
  - Acceso global: los mismos métodos actúan en objetos y en cualquier parte del sistema.
  - Seguridad global: autenticación y acceso uniformes en todo el sistema.
  - Disponibilidad global: funcionamiento correcto en presencia de fallos parciales.
  - Gestión global: posibilidad de gestión centralizada del sistema.

- Características funcionales:
  - Cada usuario o cliente trabaja con su terminal local inteligente, con lo que se obtienen mejores tiempos de respuesta.
  - Los recursos necesarios que no estén disponibles sobre el terminal local (ordenador personal o estación de trabajo) pueden tomarse del ordenador central a través de la red de telecomunicaciones.

- Características físicas:
  - Sistemas informáticos distribuidos en los que lo ordenadores, a través de la organización, están conectados por medio de una red de telecomunicaciones.
  - Cada ordenador sobre la red tiene capacidad de tratamiento autónomo que permite servir las necesidades de los usuarios locales.
  - También proporciona acceso a otros elementos de la red o a servidores centrales.
  - Toma importancia la red de comunicaciones de datos.

- Características lógicas:
  - Cada tarea individual puede ser analizada para determinar si puede distribuirse o no. En general, las tareas más complejas o de carácter estratégico para la organización se mantienen en el ordenador central. Las tareas de complejidad media o específicas para un determinado grupo de usuarios se distribuyen entre las máquinas locales de ese grupo.
  - La plataforma física seleccionada puede ajustarse a las necesidades del grupo de usuarios, con lo que surgen los ordenadores especializados para determinados tipos de tareas.

- Ventajas:
  - Funcionamiento autónomo de los sistemas locales, lo que origina un buen tiempo de respuesta.
  - Los sistemas de información llegan a todos los departamentos de la empresa.
  - Abre posibilidades de trabajo mucho más flexibles y potentes.

❖ Inconvenientes:
- Requiere un intenso flujo de informaciones (muchas veces no útiles como pantallas y datos incorrectos) dentro de la red, lo que puede elevar los costos de las comunicaciones.
- Supone una mayor complejidad.
- Si los sistemas no están integrados, pueden producirse problemas de inconsistencia de datos.

2.2.3 Arquitectura Cliente/Servidor:

El concepto de cliente/servidor proporciona una forma eficiente de utilizar todos estos recursos de máquina, de tal forma que la seguridad y fiabilidad que proporcionan los entornos mainframe se traspasa a la red de área local. A esto hay que añadir la ventaja de la potencia y simplicidad de los ordenadores personales.

La arquitectura Cliente/Servidor es un modelo para el desarrollo de sistemas de información, en el que las transacciones se dividen en procesos independientes que cooperan entre sí para intercambiar información, servicios y/o recursos.

Se denomina cliente al proceso que inicia el diálogo o solicita los recursos. Se llama servidor al proceso que responde a las solicitudes.

Es el modelo de interacción más común entre aplicaciones en una red. No forma parte de los conceptos de Internet como los protocolos IP, TCP o UDP; sin embargo, todos los servicios estándares de alto nivel propuestos en Internet funcionan según este modelo.

Los principales componentes del esquema Cliente/Servidor son:
- Los Clientes.
- Los Servidores.
- La infraestructura de comunicaciones.

En este modelo, las aplicaciones se dividen de forma que el servidor contiene la parte que debe ser compartida por varios usuarios y en el cliente permanece sólo lo particular de cada usuario.

Los Clientes interactúan con el usuario, usualmente de forma gráfica. Frecuentemente se comunican con procesos auxiliares que se encargan de establecer conexión con el servidor, enviar lo pedido, recibir la respuesta, manejar los fallos y realizar las actividades de sincronización y de seguridad.

Los Clientes realizan, generalmente, funciones como:
- Manejo del interfaz del usuario.
- Captura y validación de los datos de entrada.
- Generación de consultas e informes sobre las bases de datos.

Los Servidores proporcionan un servicio al cliente y devuelven los resultados. En algunos casos, existen procesos auxiliares que se encargan de recibir solicitudes del cliente, verificar la protección, activar un proceso servidor para satisfacer el pedido,

recibir su respuesta y enviarla al cliente, etc. Además, deben manejar los interbloqueos, la recuperación ante fallos y otra serie de aspectos afines.

Por las razones anteriores, la plataforma computacional asociada con los servidores es más poderosa que la de los clientes. Por ello, se utilizan PC´s poderosos, estaciones de trabajo, microcomputadoras o sistemas grandes. Además deben manejar servicios como la administración de la red, mensajes, control y administración de la entrada al sistema (login), auditoría, recuperación, contabilidad, etc. Usualmente, en los servidores existe algún tipo de Bases de Datos.

Por su parte, los servidores realizan, entre otras, las siguientes funciones:

- Gestión de periféricos compartidos.
- Control de accesos concurrentes a bases de datos compartidas.
- Enlaces de comunicaciones con otras redes de área local o extensa (LAN o WAN).

Siempre que un Cliente requiere un servicio lo solicita al Servidor correspondiente y éste le responde con el servicio solicitado. Normalmente, aunque no necesariamente, el cliente y el servidor están ubicados en distintos procesadores. Los Clientes se suelen situar en ordenadores personales y/o estaciones de trabajo, mientras que los Servidores se sitúan en procesadores departamentales o de grupo.

Las aplicaciones y datos críticos se hallan habitualmente en servidores redundantes o en sistemas de alta disponibilidad sin puntos de fallo únicos.

Para que los Clientes y los Servidores se puedan comunicar, se requiere una infraestructura de comunicaciones, la cual proporciona los mecanismos básicos de direccionamiento y transporte. La mayoría de los sistemas Cliente/Servidor actuales se basan en redes locales (LAN) y por tanto utilizan protocolos no orientados a conexión, lo cual implica que las aplicaciones deben hacer las verificaciones. La red debe tener características adecuadas de desempeño, confiabilidad, transparencia y administración.

Entre las principales características de la arquitectura Cliente/Servidor, se pueden destacar las siguientes:

- El servidor presenta a todos sus clientes un interfaz único y bien definido.
- El cliente no necesita conocer la lógica del servidor, sólo su interfaz externo.
- El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.
- Los cambios en el servidor implican pocos o ningún cambio en el cliente.
- Un único servidor típicamente sirve a una multitud de clientes, ahorrando a cada uno de ellos el problema de tener la información instalada y almacenada localmente.

Los sistemas Cliente/Servidor pueden ser de muchos tipos, dependiendo de las aplicaciones que el servidor pone a disposición de los clientes. Entre otros, existen:

- Servidores de Impresión, mediante los cuales los usuarios comparten impresoras.
- Servidores de Archivos, a través de los cuales los clientes comparten discos duros.
- Servidores de Bases de Datos, donde existe una única base de datos.
- Servidores de Lotus Notes, que permite el trabajo simultáneo de distintos clientes con los mismos datos, documentos o modelos.

Los Servidores Web también utilizan la tecnología Cliente/Servidor, aunque añaden aspectos nuevos y propios a la misma.

Las transacciones de ordenador que usan el modelo cliente/servidor son muy comunes. Por ejemplo, para revisar nuestra cuenta de banco desde nuestro ordenador, un programa cliente en nuestra máquina envía nuestra solicitud a un programa servidor en el banco. Ese programa puede, a su vez, reexpedir la solicitud a su propio programa cliente que envía una solicitud a un servidor de base de datos en otro ordenador del banco para obtener nuestro saldo. El saldo es enviado al cliente del banco, que a su vez se lo sirve de vuelta al cliente en nuestro ordenador personal, que despliega la información para que la veamos.

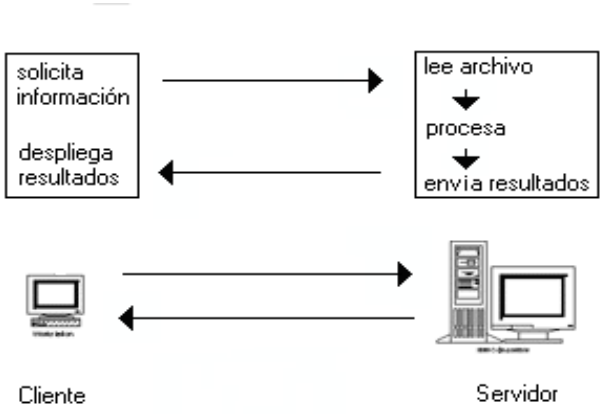
El modelo cliente/servidor se ha convertido en una de las ideas centrales de la computación en red. La mayoría de las aplicaciones de negocios que se están escribiendo actualmente usan el modelo cliente/servidor. Igual lo hace el principal protocolo de Internet, el TCP/IP.

En el modelo usual cliente/servidor, un servidor, a veces llamado daemon, se activa y espera las solicitudes de los clientes. Habitualmente, programas cliente múltiples comparten los servicios de un programa servidor común. Tanto los programas cliente como los servidores son con frecuencia parte de un programa o aplicación mayores. Con relación a Internet, nuestro navegador de la Red es un programa cliente que solicita servicios (el envío de páginas web o archivos) de un servidor web (que técnicamente se llama un servidor de Protocolo de Transporte de Hipertexto, Hypertext Transport Protocol o HTTP) en otro ordenador en algún lugar de Internet. De modo similar, nuestro ordenador con TCP/IP instalado nos permite hacer solicitudes de cliente para recibir archivos de servidores de Protocolo de Transferencia de Archivos (File Transfer Protocol, FTP) en otros ordenadores de Internet.

Otros modelos de relación entre programas incluyen el maestro/esclavo, en el que un programa está a cargo de todos los demás programas de igual a igual, donde cualquiera de los programas puede iniciar una transacción.

En el sentido más estricto, el término cliente/servidor describe un sistema en el que una máquina cliente solicita a una segunda máquina llamada servidor que ejecute una tarea específica. El cliente suele ser una computadora personal común conectada a una LAN, y el servidor es, por lo general, una máquina anfitriona, como un servidor de archivos PC, un servidor de archivos de UNIX, o una macrocomputadora o computadora de rango medio.

El programa cliente cumple dos funciones distintas: por un lado, gestiona la comunicación con el servidor, solicita un servicio y recibe los datos enviados por aquél. Por otro, maneja el interfaz con el usuario: presenta los datos en el formato adecuado y brinda las herramientas y comandos necesarios para que el usuario pueda utilizar las prestaciones del servidor de forma sencilla. El programa servidor, en cambio, básicamente sólo tiene que encargarse de transmitir la información de forma eficiente. No tiene que atender al usuario. De esta forma, un mismo servidor puede atender a varios clientes al mismo tiempo. Algunas de las principales LAN cliente/servidor con servidores especializados que pueden realizar trabajos para clientes incluyen a Windows NT, NetWare de Novell, VINES de Banyan y LAN Server de IBM entre otros. Todos estos sistemas operativos de red pueden operar y procesar solicitudes de aplicaciones que se ejecutan en clientes mediante el procesamiento de las solicitudes.



2.2.3.1 Sistemas Operativos enlazados con el Cliente/Servidor:

- NetWare de Novell:

El enfoque de Novell de servicio al usuario de LAN es único, ya que ha elegido concentrar esfuerzos en la producción de software que funciona en el hardware de redes de otros fabricantes. NetWare funciona en prácticamente cualquier IBM o compatible, y opera en todo el hardware de los fabricantes más importantes de LAN incluyendo los productos de Apple Macintosh y ARCnet. La filosofía de Novell es convertirse en un estándar de la industria por medio del dominio del mercado. El sistema operativo de red de Novell, NetWare, puede funcionar en varias topologías diferentes. Dependiendo del hardware que se seleccione, NetWare puede ejecutarse en una red configurada como estrella, agrupamiento de estrellas, Token Ring e incluso en un bus.

NetWare está diseñado para ofrecer un verdadero soporte de servidor de archivos de red. En el modelo OSI, el software de servidor de archivos de Novell reside en la capa de aplicaciones, mientras que el software operativo de disco (DOS) reside en la capa de presentación. El software de servidores de archivos forma una cubierta alrededor de los sistemas operativos, como el DOS, y es capaz de interceptar comandos de programas de aplicaciones antes de que lleguen al procesador de comandos del sistema operativo. El usuario de las estaciones de trabajo no se da cuenta de este fenómeno; simplemente pide un archivo de datos o un programa sin preocuparse acerca de dónde está ubicado.

NetWare permite que el supervisor defina el acceso a directorios. El administrador del sistema puede determinar que un programa o archivo sea compartible o no compartible. NetWare también contiene una función predeterminada de bloqueo de archivos, lo cual significa que los programas de un solo usuario podrían ser utilizados por diferentes usuarios, pero sólo uno a la vez.

Si un archivo no es compartible, los diferentes usuarios pueden ver el archivo en el modo de sólo lectura, pero no pueden escribir en él mientras otro usuario lo esté utilizando en modo de lectura o escritura. Los programas o archivos que se designan como compartibles con capacidad de bloqueo de registros operan en modo multiusuario real: varios usuarios pueden leer y escribir en ellos en forma simultánea, siempre que sólo un usuario escriba en un registro específico en un momento determinado.

NetWare requiere que se tenga acceso a los directorios de la red a través de unidades de red específicas. Las unidades de red apuntan a directorios de la red y no a unidades físicas de discos.

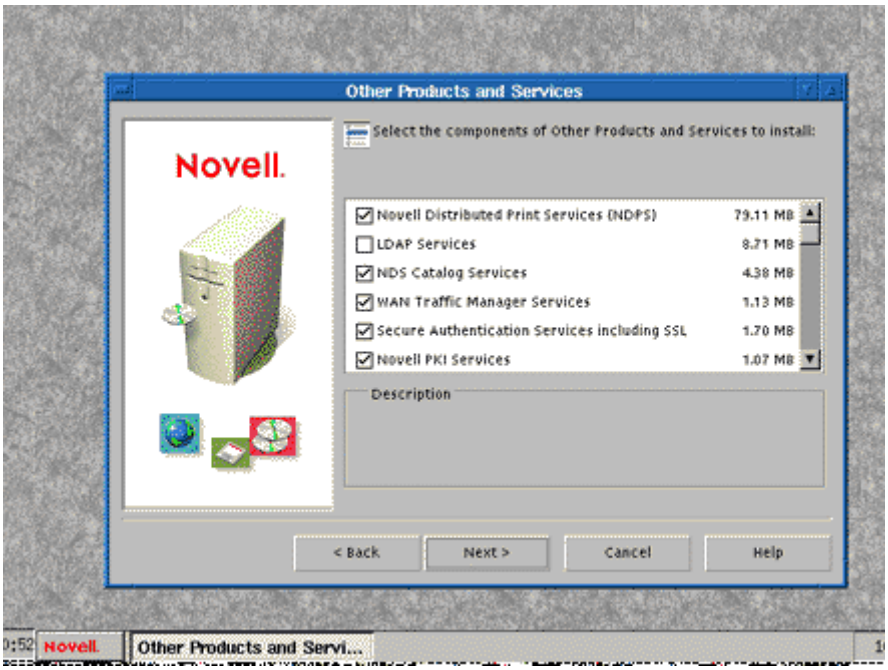
Cada estación de trabajo puede asignar 21 letras de unidades lógicas (de la F a la Z). Esto supone que DOS utiliza las letras predeterminadas de la A a la E. Las unidades de búsqueda de NetWare permiten que el sistema operativo localice los archivos de programas en directorios diferentes del directorio predefinido correspondiente. Al colocar los programas que se usan universalmente en directorios de acceso público y luego mapearlos en una unidad de búsqueda, el servidor de archivos localiza los programas solicitados aún si no se encuentran en el directorio actual desde donde se hace la solicitud.

NetWare de Novell ofrece los sistemas de seguridad más importantes del mercado, ya que proporciona seguridad de servidores de archivos en cuatro formas diferentes:

- Procedimiento de registro de entrada.
- Derechos encomendados.
- Derechos en directorio.
- Atributos de archivo.

NetWare 4.x es un sistema operativo de red de Novell diseñado para la computación empresarial. Puede manejar hasta 1.000 usuarios en un solo servidor. Está basado en la arquitectura de 32 bits del rango que va de los procesadores 386 al Pentium. Este sistema operativo es similar al sistema operativo de la versión 3.12, aunque tiene mejoras significativas. La característica principal de 4.x son los Servicios de Directorios de NetWare (NetWare Directory Services, NDS). NDS es un amplio servicio empresarial que enlaza a los servidores de archivos con los recursos de la red, como las impresoras, en un directorio jerárquico orientado a objetos. Es una base de datos distribuida globalmente que proporciona un solo punto de registro y que está construida para facilitar la partición y réplica de todos los servidores. Diseñado para redes grandes, NetWare 4.x permite que un administrador de red maneje docenas de servidores de archivos desde una sola consola. También ofrece comunicaciones remotas mejoradas.

Una de las características de seguridad importante implantada en la versión 4.x es la autenticación de paquetes, para evitar que intrusos capturen paquetes y falsifiquen una identificación de sesión de un usuario para conseguir privilegios de acceso.



Novell cree que la industria de las computadoras está ahora en una segunda etapa de conectividad LAN, en la cual las LAN se conectan a computadoras de rango medio y macrocomputadoras mediante compuertas o interfaces directos. Durante los últimos años, Novell ha planeado una arquitectura que sea consistente con un futuro caracterizado por una creciente conectividad, flujo de información entre computadoras grandes y pequeñas, y compatibilidad entre múltiples fabricantes. El plan de Novell, conocido como Arquitectura Universal de Red, es dirigirse hacia una arquitectura que abarque cualquier plataforma.

❖ Windows NT Server de Microsoft:

Windows NT de Microsoft es un verdadero sistema operativo de 32 bits muy poderoso, que está disponible en versiones cliente y servidor. Entre las características clave de NT está la multitarea prioritaria, procesos de multilectura o hebras, portabilidad y soporte para multiprocesamiento simétrico. La multitarea prioritaria permite la realización de múltiples tareas preferentes y subordinadas. Es NT y no los programas específicos quien determina cuándo deberá interrumpirse un programa y empezar a ejecutar otro. Procesos de lectura múltiple o hebras es un término que, en NT, se refiere a los hilos que funcionan como agentes de ejecución. Tener hebras de ejecución múltiple dentro de un mismo proceso significa que un proceso ejecuta, de manera simultánea, diferentes partes de un programa en diferentes procesadores.

El multiprocesamiento simétrico permite que los requerimientos de sistema y aplicación se distribuyan de manera uniforme entre todos los procesadores disponibles, haciendo que todo funcione mucho más rápido. Windows NT emplea el sistema de archivos NT (NTFS). Este sistema de archivos soporta nombres de archivo de hasta 256 caracteres. También permite el rastreo de transacciones. Esto significa que si el sistema falla, NT regresa los datos al estado inmediato anterior a la caída del sistema.

Microsoft diseñó Windows NT para que fuera portátil. Está compuesto de un kernel o núcleo, así como de diferentes subsistemas del sistema. Hay subsistemas disponibles para aplicaciones que ejecutan programas basados en OS/2 y POSIX . Un procesador DOS virtual (VDM) ejecuta MS-DOS y aplicaciones Windows de 16 bits. NT incluye software de red de punto a punto para que los usuarios de NT puedan compartir archivos y aplicaciones con otros usuarios que ejecuten NT o Windows para Trabajo en Grupo.

En temas de seguridad, NT posee los siguientes aspectos: requiere que los usuarios introduzcan una contraseña cada vez que inician el sistema operativo, estén o no conectados a un servidor. Cada vez que se inicia NT, éste solicita una contraseña. NT califica para la certificación gubernamental C-2 para ambientes seguros. Microsoft ha señalado que en el futuro ofrecerá mejoras que elevarán el nivel de seguridad de NT y lo harán aún más atractivo para las dependencias del gobierno.

Una función de seguridad de NT es el administrador de usuarios. Este programa garantiza que las contraseñas se sujeten a la política de la compañía. También permite que cada máquina NT sea configurada para cierto número de usuarios, dando a cada uno de ellos su propio nivel de privilegios. Además, es posible crear grupos y dar los mismos privilegios a todos los integrantes de un grupo.

Otra función de seguridad clave es el visor de eventos. Este programa le permite a los administradores de red visualizar una bitácora de todos los errores e infracciones a la red, incluyendo la hora, fecha y tipo de infracción, así como el lugar donde ocurrió el evento y el nombre del usuario implicado.

Windows NT Server ofrece compartición de archivos integrada, capacidad de compartición de impresoras para la computación en grupos de trabajo y un interfaz de sistema de red abierto, que incluye soporte integrado para IPX/SPX, TCP/IP, NetBEUI y otros transportes. NT Server es compatible con redes existentes como VINES, NetWare, UNIX, LAN Manager 2.x y Windows para Trabajo en Grupo. Windows NT incluye interfaces de programación de aplicación (API) que permiten que los fabricantes de sistemas operativos de red (NOS) escriban software de cliente para que sus productos puedan ejecutarse con éste. NT da soporte a clientes Macintosh y los trata de la misma manera como usuarios de la red, dando soporte al protocolo de archivo AppleTalk. Los usuarios de Macintosh pueden acceder el servidor NT Server como si se tratara de un servidor AppleShare.

- LAN Server de IBM:

LAN Server es un sistema operativo de red que se ejecuta bajo OS/2. Este software de servidor de archivos proporciona lo que IBM llama “relaciones solicitador / servidor” (y lo que el resto de la industria conoce como relaciones cliente/servidor).

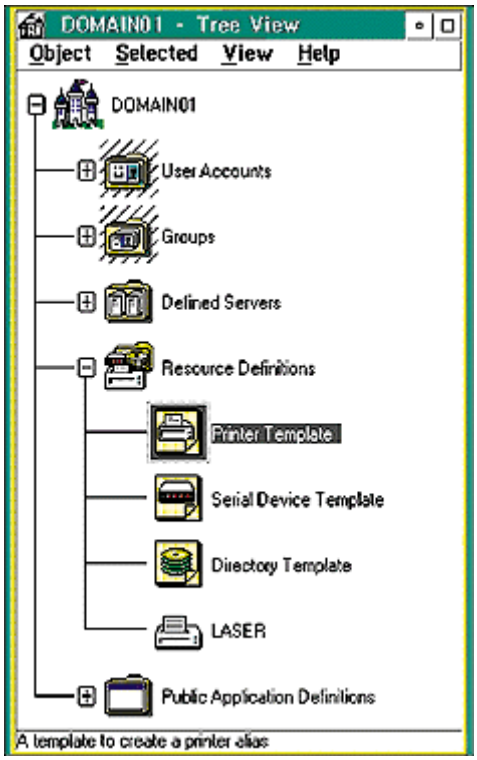
LAN Server ofrece funciones de acceso a bases de datos mejoradas debido a la disponibilidad del componente Servicios de Conexión de Bases de Datos Distribuidas/2 (DDCS/2), que forma parte de manera opcional en la arquitectura de sistemas de red de IBM. Esta característica permite conexiones entre las bases de datos anfitrionas y las bases de datos ubicadas en estaciones remotas clientes de red.

El concepto de dominio es muy importante para entender el funcionamiento de una red LAN Server. Un grupo de estaciones de trabajo y uno o más servidores constituyen el dominio. Un usuario que cuente con un ID (Identificación) de usuario para el dominio puede registrarse en él desde una estación de trabajo solicitadora y acceder a los recursos de dicho dominio.

LAN Server exige la creación de un dominio como mínimo y que haya un servidor que actúe como controlador del dominio. Se puede tener otro servidor que actúe como controlador de dominio de respaldo.

Una función muy valiosa de LAN Server, conocida como independencia de ubicación, le permite al administrador de red tratar un grupo de servidores de red como si fueran un solo servidor.

A diferencia de otros sistemas operativos de red, este programa utiliza el poder de OS/2 para rastrear la actividad de la red y emitir alertas. El interfaz del usuario basado en gráficos es consistente con la arquitectura de aplicaciones del sistema de IBM, lo cual constituye el plan a largo plazo de esta compañía para ofrecer un interfaz uniforme en toda su línea de productos. LAN Server es preferible sobre otros sistemas operativos de red para aquellos clientes que tienen una gran inversión en equipos de macrocomputadoras por medio de su Administrador de Comunicaciones y un mejor acceso a bases de datos de macrocomputadora con su rango DB2/2 y DDCS/2.



2.2.3.2 Componentes esenciales:

Una infraestructura Cliente/Servidor consta de tres componentes esenciales, todos ellos de igual importancia y estrechamente ligados.

- **Plataforma Operativa.** La plataforma deberá soportar todos los modelos de distribución Cliente/Servidor, todos los servicios de comunicaciones y deberá utilizar, preferentemente, componentes estándar de la industria para los servicios de distribución. Los desarrollos propios deben coexistir con las aplicaciones estándar y su integración deberá ser imperceptible para el usuario. Igualmente, podrán acomodarse programas escritos utilizando diferentes tecnologías y herramientas.

- **Entorno de Desarrollo de Aplicaciones.** Debe elegirse después de la plataforma operativa, aunque es conveniente evitar la proliferación de herramientas de desarrollo. Se garantizará que el enlace entre éstas y el middleware no sea excesivamente rígido. Será posible utilizar diferentes herramientas para desarrollar partes de la aplicación. Un entorno de aplicación incremental debe posibilitar la coexistencia de procesos de cliente y servidor desarrollados con distintos lenguajes de programación y/o herramientas, así como utilizar distintas tecnologías (por ejemplo, lenguaje procedural, lenguaje orientado a objetos, multimedia), y que han sido puestas en explotación en distintos momentos del tiempo.

- **Gestión de Sistemas.** Estas funciones aumentan considerablemente el costo de una solución, pero no se pueden evitar. Siempre deben adaptarse a las necesidades de la organización y al decidir la plataforma operativa y el entorno de desarrollo, es decir, en las primeras fases de la definición de la solución merece la pena considerar los aspectos siguientes:
  - ¿Qué necesitamos gestionar?
  - ¿Dónde estarán situados los procesadores y estaciones de trabajo?
  - ¿Cuántos tipos distintos se soportarán?
  - ¿Qué tipo de soporte es necesario y quién lo proporciona?

2.2.3.3 Ventajas e inconvenientes de la arquitectura Cliente/Servidor:

- Ventajas:
  - *Aumento de la productividad:*
    - Los usuarios pueden utilizar herramientas que le son familiares, como hojas de cálculo y herramienta de acceso a bases de datos.
    - Mediante la integración de las aplicaciones Cliente/Servidor con las aplicaciones personales de uso más habitual, los usuarios pueden construir soluciones particularizadas que se ajusten a sus necesidades cambiantes.
    - Un interfaz gráfico de usuario consistente que reduce el tiempo de aprendizaje de las aplicaciones.

  - *Menores costes de operación:*
    - La existencia de plataformas hardware cada vez más baratas. Esto constituye a su vez una de las más palpables ventajas de este esquema, la posibilidad de utilizar máquinas considerablemente más baratas que las requeridas por una solución centralizada basada en sistemas grandes.

- Permiten un mejor aprovechamiento de los sistemas existentes, protegiendo la inversión. Por ejemplo, la compartición de servidores (habitualmente caros) y dispositivos periféricos (como impresoras) entre máquinas clientes permite un mejor rendimiento del conjunto.
- Se pueden utilizar componentes, tanto de hardware como de software, de varios fabricantes, lo cual contribuye considerablemente a la reducción de costes y favorece la flexibilidad en la implantación y actualización de soluciones.
- Proporcionan un mejor acceso a los datos. El interfaz de usuario ofrece una forma homogénea de ver el sistema, independientemente de los cambios o actualizaciones que se produzcan en él y de la ubicación de la información.
- El movimiento de funciones desde un ordenador central hacia servidores o clientes locales origina el desplazamiento de los costes de ese proceso hacia máquinas más pequeñas y, por tanto, más baratas.

- *Mejora en el rendimiento de la red:*
  - Las arquitecturas cliente/servidor eliminan la necesidad de mover grandes bloques de información por la red hacia ordenadores personales o estaciones de trabajo para su proceso. Los servidores controlan los datos, procesan peticiones y después transfieren sólo los datos requeridos a la máquina cliente. Entonces, la máquina cliente presenta los datos al usuario mediante interfaces amigables. Todo esto reduce el tráfico de la red, lo que facilita que pueda soportar mayor número de usuarios.
  - Se pueden integrar PC´s con sistemas medianos y grandes, sin que todas las máquinas tengan que utilizar el mismo sistema operacional.
  - Si se usan interfaces gráficos para interactuar con el usuario, el esquema Cliente/Servidor presenta la ventaja, con respecto a uno centralizado, que no es siempre necesario transmitir información gráfica por la red, pues ésta puede residir en el cliente, lo cual permite aprovechar mejor el ancho de banda de la red.
  - Tanto el cliente como el servidor pueden escalar para ajustarse a las necesidades de las aplicaciones. Los microprocesadores utilizados en los respectivos equipos pueden dimensionarse a partir de las aplicaciones y el tiempo de respuesta que se requiera.
  - La existencia de varios microprocesadores proporciona una red más fiable. Un fallo en uno de los equipos no significa necesariamente que el sistema deje de funcionar.
  - En una arquitectura como ésta, los clientes y los servidores son independientes los unos de los otros, con lo que pueden renovarse para aumentar sus funciones y capacidad de forma independiente, sin afectar al resto del sistema.
  - La arquitectura modular de los sistemas Cliente/Servidor permite el uso de ordenadores especializados (servidores de bases de datos, servidores de ficheros, estaciones de trabajo para CAD, etc.).
  - Permite centralizar el control de sistemas que estaban descentralizados como, por ejemplo, la gestión de los ordenadores personales que antes estuvieron aislados.

- ♦ Es más rápido el mantenimiento y el desarrollo de aplicaciones, puesto que se pueden emplear las herramientas existentes (por ejemplo los servidores SQL o las herramientas de más bajo nivel como los sockets o las RPC).
- ♦ El esquema Cliente/Servidor contribuye, además, a proporcionar a las diferentes direcciones de una institución soluciones globales, pero permitiendo además la integración de la información a nivel global.

❖ Inconvenientes:

- Hay una alta complejidad tecnológica al tener que integrar una gran variedad de productos.
- Por otra parte, el mantenimiento de los sistemas es más difícil pues implica la interacción de diferentes partes de hardware y de software, distribuidas por distintos proveedores, lo cual dificulta el diagnóstico de fallos.
- Requiere un fuerte diseño de todos los elementos involucrados en los sistemas de información (modelo de datos, procesos, interfaces, comunicaciones, almacenamiento de datos, etc.). Además, en la actualidad, existen pocas herramientas que ayuden a determinar la mejor forma de dividir las aplicaciones entre la parte cliente y la parte servidor.
- Es importante que los clientes y los servidores utilicen el mismo mecanismo (por ejemplo sockets o RPC), lo cual implica que deben de tener los mismos mecanismos generales que existen en diferentes plataformas.
- Se cuenta con escasas herramientas para la administración y ajuste del desempeño de los sistemas.
- Es más difícil asegurar un elevado grado de seguridad en una red de clientes y servidores que en un sistema con un único ordenador centralizado. Se deben hacer verificaciones en el cliente y en el servidor. También se puede recurrir a otras técnicas como el cifrado.
- Un aspecto directamente relacionado con el anterior es el de cómo distribuir los datos en la red. En el caso de una empresa, por ejemplo, éste puede ser hecho por departamentos, geográficamente, o de otras maneras. Además, hay que tener en cuenta que, en algunos casos por razones de confiabilidad o eficiencia, se pueden tener datos replicados y que puede haber actualizaciones simultaneas.
- A veces, los problemas de congestión de la red pueden degradar el rendimiento del sistema por debajo de lo que se obtendría con una única máquina (arquitectura centralizada). También el interfaz gráfico de usuario puede, a veces, ralentizar el funcionamiento de la aplicación.
- Existen multitud de costes ocultos que encarecen su implantación (formación en nuevas tecnologías, licencias, cambios organizativos, etc.).

2.2.4 Comunicación Cliente/Servidor:

En el modelo Cliente/Servidor, los procesos llamados servidores proporcionan servicios a los clientes en una red.

La mayor parte de las redes de área local (LAN) cuentan con servidores de archivos que gestionan el espacio de disco común, facilitando el compartimiento de archivos y la preparación de copias de respaldo.

Muchas aplicaciones y servicios de red como el correo, la transferencia de archivos (FTP, File Transfer Protocol), la verificación de autenticidad (Kerberos), el ingreso remoto (telnet) y el acceso a sistemas de archivos remotos (NFS) se basan en el modelo Cliente/Servidor.

El servidor puede estar en la misma máquina que el cliente o en una distinta, en cuyo caso la comunicación se realizará a través de red.

Dos clases de protocolos de comunicación de bajo nivel que manejan el paradigma de Cliente/Servidor son:

- Los protocolos orientados a las conexiones.
- Los protocolos sin conexiones.

En el modelo de comunicación orientado a las conexiones, un servidor espera que un cliente solicite una conexión. Una vez que establece ésta, la comunicación se realiza utilizando asas (descriptores de ficheros) y la dirección del servidor no se incluye en los mensajes del usuario.

Los protocolos orientados a conexiones tienen un gasto extra de establecimiento. Una estrategia alternativa es utilizar un protocolo sin conexiones. El cliente envía un solo mensaje a un servidor; el servidor presta el servicio y devuelve una respuesta.

Los protocolos, tanto aquellos sin conexiones como los orientados a las conexiones, se consideran de bajo nivel en cuanto a que la solicitud de servicio implica una comunicación visible. El programador está consciente de la existencia y la ubicación del servidor y debe nombrar explícitamente el servidor específico al que desea acceder.

La asignación de nombres a los servidores en un entorno de red es un problema difícil. El método obvio consiste en designar un servidor por su ID de proceso y su ID de nodo. Sin embargo, como el ID de proceso se asigna cronológicamente en el momento en el que el proceso inicia su ejecución, es difícil saber con antelación el ID del proceso de un proceso específico de un nodo.

La convención de asignación de nombres más común consiste en los números enteros pequeños llamados puertos. Un servidor escucha en un puerto bien conocido (Well Kown Ports) que se ha asignado previamente para un servicio particular. Al establecer la conexión, el cliente especifica claramente una dirección de nodo y un número de puerto de ese nodo.

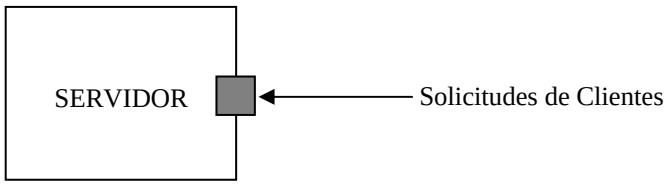
También se da la asignación de nombres de nivel intermedio mediante llamadas a procedimientos remotos (RPC, remote procedure calls). Aunque éstas también requieren la especificación clara del nodo, permiten al usuario solicitar un servicio determinado en el nodo por nombre, sin tener que especificar el número de puerto.

Uno de los objetivos de los sistemas operativos distribuidos es proporcionar un interfaz unificado entre el usuario y los servicios de sistema en toda la red. En una situación ideal, más de un servidor en la red podría prestar un servicio determinado.

Sería cómodo solicitar el servicio por nombre y dejar que el sistema designase el servidor. Desafortunadamente, este tipo de interfaz de alto nivel todavía es difícil de conseguir.

2.2.4.1 Estrategias Cliente/Servidor:

La comunicación Cliente/Servidor más sencilla se efectúa a través de un solo puerto de comunicación como el que se muestra en la siguiente figura.



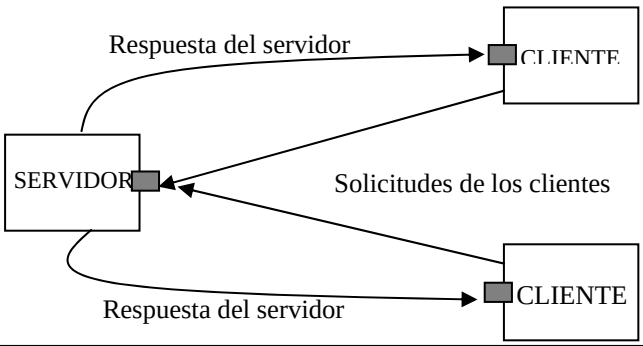
Si el cliente y el servidor comparten un sistema de archivos y se están ejecutando en la misma máquina, el puerto único pude ser FIFO (primero que entra, primero que sale). En una red, ese puerto puede ser un socket o una conexión TLI.

Cuando el servidor se inicia, abre su FIFO bien conocido (o su conexión de socket con un puerto bien conocido) y espera solicitudes de los clientes. Cuando un cliente necesita un servicio, abre el FIFO (o una conexión de socket con el puerto bien conocido del servidor) y escribe su solicitud. A continuación, el servidor da el servicio.

Este enfoque funciona muy bien si sólo hay un cliente y no requiere una respuesta. Si hay más de un cliente, es necesario establecer una convención para enviar el ID del proceso del cliente a fin de poder distinguir las solicitudes de cada uno de los clientes. Desde luego, sería conveniente que el mecanismo de identificación impidiera que los clientes falsificaran el ID de otro usuario al pretender hacerlo.

Un solo puerto no es suficiente en caso de que el cliente necesite recibir una respuesta del servidor, pues no se cuenta con un mecanismo para que el cliente obtenga la respuesta correcta.

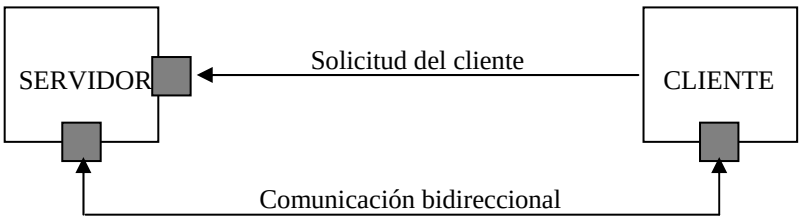
Supongamos que el servidor sólo cuenta con un único FIFO para las respuestas. Puesto que los elementos se sacan del FIFO cuando se leen, y no hay nada que impida a un cliente leer la respuesta dirigida a otro, cada uno necesita su propio canal para las respuestas del servidor, como aparece en la siguiente figura.



En la implementación FIFO, el servidor puede abrir un FIFO específico para un cliente añadiendo el ID de proceso del cliente al nombre del FIFO de respuesta. El cliente envía una solicitud que incluye su ID (no falsificable) y abre un FIFO convenido para leer la respuesta del servidor. El servidor debe asegurarse de crear semejante FIFO de respuesta con los permisos apropiados de modo que un cliente, aunque lo intente, no pueda robar las respuestas dirigidas a otro.

La implementación con sockets cuenta con una llamada *recvfrom* que permite al servidor escuchar en un puerto de socket bien conocido para detectar solicitudes. Cada solicitud incluye la identidad de quién la envió. El servidor simplemente utiliza esta identificación en una respuesta *sendto* al cliente. Las llamadas *recvfrom* (recibir de) y *sendto* (enviar a) constituyen la base del protocolo de sockets sin conexiones.

Si el cliente y el servidor necesitan seguir interactuando durante el procesamiento de la solicitud, resulta útil contar con un canal de comunicación bidireccional que sea privado pero que no requiera un intercambio de información de ID de proceso en cada mensaje.



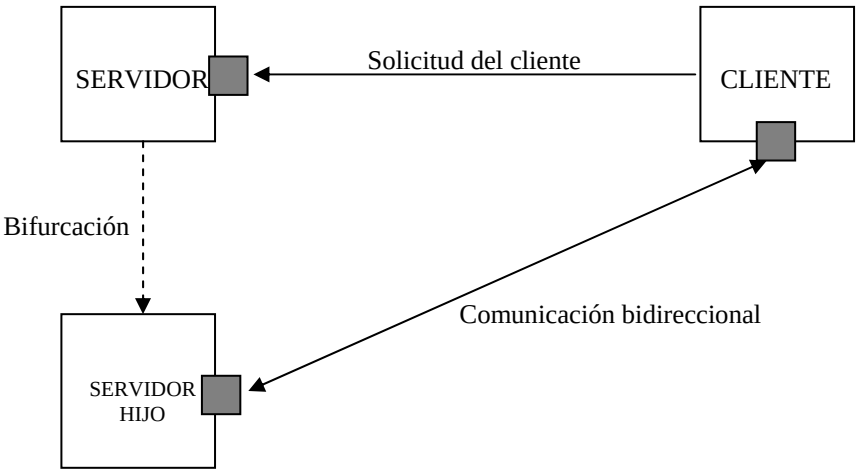
En esta figura se ilustra un mecanismo de transferencia en el que la solicitud inicial del cliente sólo sirve para establecer el canal de comunicación bidireccional que es privado pero no específico para un cliente. El canal es privado porque no es accesible a otros procesos y no es específico para un cliente porque no se identifica con el ID de proceso del cliente. Los protocolos orientados a conexión utilizan mecanismos de transferencia para establecer un canal de comunicación entre el cliente y el servidor.

Una vez que un servidor recibe una solicitud y establece un canal de comunicación puede adoptar varias estrategias distintas para atender las solicitudes. Una posibilidad es que cuando un servidor recibe una solicitud se dedique íntegramente a atender esa solicitud antes de aceptar solicitudes adicionales.

En la figura anterior se representaba la estrategia de implementación de un servidor en serie. El siguiente pseudocódigo ilustra la estrategia de un servidor en serie.

```
For ( ; ; )
        {
                escuchar solicitud de cliente
                crear canal privado de comunicación bidireccional
                while (no hay error en el canal de comunicación)
                        leer solicitud del cliente
                        atender solicitud y responder al cliente
                cerrar el canal de comunicación
        }
```

Un servidor ocupado que atiende solicitudes de larga duración, como transferencia de ficheros, no puede utilizar la estrategia de servidor en serie porque sólo permite atender a una solicitud a la vez. En la estrategia de servidor padre, el servidor bifurca un hijo que presente el servicio real al cliente mientras el servidor vuelve a escuchar para detectar solicitudes adicionales. En la siguiente figura se representa la estrategia de servidor padre y es ideal para servicios como la transferencia de archivos que tardan un tiempo relativamente largo e implican gran cantidad de bloqueos.

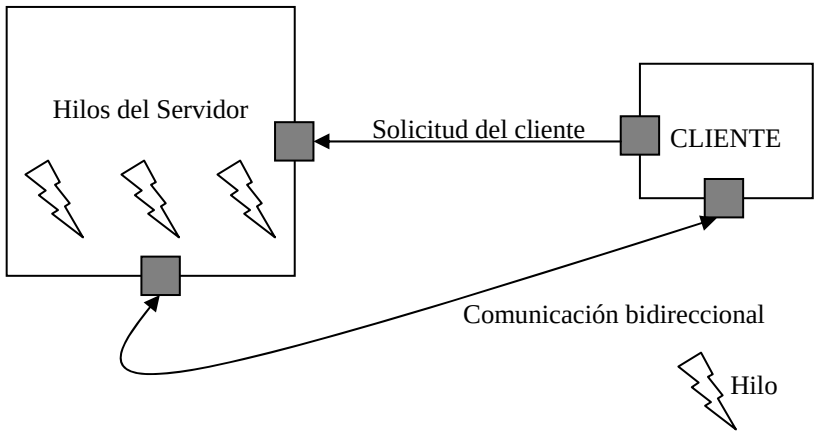


El siguiente pseudocódigo ilustra la estrategia de servidor padre.

```
For (; ; )
        {
                escuchar solicitud cliente
                crear canal privado de comunicación bidireccional
                bifurcar un hijo que atienda la solicitud
                cerrar el canal de comunicación
                eliminar "zombies"
        }
```

Puesto que el servidor hijo se encarga de presentar el servicio real en la estrategia de servidor padre, el servidor puede aceptar múltiples solicitudes de clientes en rápida sucesión. La estrategia es análoga a la de los anticuados tableros de conmutador telefónico que todavía hay en algunos hoteles. Un cliente llama al número principal del hotel (solicitud de conexión). El operador del tablero (servidor) contesta la llamada, puentea la conexión a la habitación apropiada (el servidor hijo), se desliga de la conversación y continúa escuchando para detectar llamadas adicionales.

La estrategia de servidor con hilos es una alternativa con bajo gasto extra que puede utilizarse en lugar de la estrategia de servidor padre. Esta estrategia queda representada en la siguiente figura.



En lugar de bifurcar un hijo para que atienda la solicitud, el servidor crea un hilo en su propio espacio de proceso. Los hilos tienen un gasto extra mucho menor y el enfoque pueden ser muy eficiente, sobre todo si la solicitud es pequeña o en ella predomina la Entrada/Salida.

Una desventaja de la estrategia de servidor con hilos es la posible interferencia entre varias solicitudes en virtud del espacio de direcciones compartido. En el caso de servicios en los que predominan los cálculos, los hilos adicionales pueden reducir la eficiencia del hilo principal del servidor o incluso bloquearlo. El diseño del servidor debe tener suficiente paralelismo a nivel de núcleo para tener un buen rendimiento.

2.2.5 Internet dentro de la arquitectura Cliente/Servidor:

2.2.5.1 Introducción al concepto Internet:

La WWW fue creada en 1989-1990 por un grupo de científicos en el CERN (Consejo Europeo para la Investigación Nuclear en Génova, *Conseil Européen pour la Recherche Nucléaire*) con un objetivo muy concreto: compartir información de manera flexible, sencilla y distribuida.

Uno de los principales responsables es *Tim Berners Lee* para quien la WWW es "*un sistema de información multimedia distribuido, heterogéneo y colaborativo*":

- Es multimedia porque maneja textos, imágenes estáticas y más recientemente imágenes dinámicas (vídeo y animación) y sonido.
- Es distribuido porque no hay un grupo puntual central donde la información está concentrada, sino que la Web cubre todo el globo con miles de servidores, cada uno conteniendo algo de información organizada como un gran hipertexto.
- Es heterogéneo porque incluye varios servicios de Internet que la precedieron y distintos del de la Web, como por ejemplo Gopher, FTP, Newsgroups, etc.

- Y, finalmente, es colaborativo porque permite que cualquiera pueda agregar nueva información y la misma esté disponible rápidamente para todos los demás.

Para cualquiera de nosotros, la Web es además una ventana a la información mundial, muy fácil de utilizar. Este factor es el responsable directo de que la Web ES FÁCIL DE USAR.

2.2.5.2 Componentes:

Algunos de los componentes fundamentales de la World Wide Web son:

- La arquitectura Cliente/Servidor: permite este tipo de operación descentralizada.
- HTTP (*Hipertext Transfer Protocol*, protocolo de transferencia de Hipertextos): permite pasar el hipertexto entre clientes y servidores.
- HTML (*Hipertext Markup Language*, protocolo de marcado de Hipertexto): permite expresar toda esa información multimedia y que el cliente la interprete para nosotros.
- Los estándares: permiten el avance gracias a los acuerdos en común.

2.2.5.3 Cliente/Servidor en la red:

Es el modelo de interacción más común entre las aplicaciones en una red. No forma parte de los conceptos de Internet como los protocolos IP, TCP o UDP; sin embargo, todos los servicios estándares de alto nivel propuestos en Internet funcionan según este modelo.

El World Wide Web es, básicamente, un sistema Cliente/Servidor:

- El programa cliente es un *browser* (navegador) que permite ver y explorar la información que hay en el WWW. Existen diversos navegadores: Mosaic, Microsoft Explorer, Netscape. El programa cliente trae la página pedida, interpreta el texto y los comandos de formato que contiene y visualiza en la pantalla la página correctamente formateada.
- En cada servidor Web hay un programa servidor escuchando las peticiones de conexiones que los clientes (generalmente browsers) le realicen. Después de efectuarse la conexión, el cliente envía una petición y el servidor envía una respuesta. Posteriormente, la conexión es liberada. El protocolo que define las peticiones y las respuestas se llama HTTP (Hipertext Transfer Protocol).

Este tipo de estrategia de diseño de programas separa las funciones en dos programas distintos:

Uno de ellos, el cliente, se especializa en pedir. Es el que normalmente tenemos en nuestras computadoras y aquel con el que interactuamos. Debido a que no tiene todas las capacidades necesarias para completar el servicio (sólo se especializa en pedir), es

más pequeño y posee menos requerimientos para correr satisfactoriamente, es decir, necesita menos memoria RAM, menos potencia de procesador, etc.

Del otro lado de la conexión se encuentra la otra mitad, el servidor. Lo único que hace es esperar pedidos de sus clientes y satisfacerlos (ambos programas pueden residir en la misma máquina).

2.2.5.4 Protocolo de Internet:

El protocolo utilizado por los clientes y servidores de la Web se denomina HTTP. Así como nosotros utilizamos el español en nuestro país, los servidores y clientes de la World Wide Web utilizan HTTP para entenderse.

Éste no es el único protocolo utilizado en la Web. Una de las características más importantes de la Web es que reúne a todos los otros servicios "bajo el mismo sombrero". Pero el HTTP es el protocolo primario y es el que fue específicamente diseñado para transferir documentos u objetos hipermedia.

El protocolo en sí define una transacción simple de cuatro pasos:

- El cliente establece una conexión con el servidor.
- El cliente hace un pedido al servidor (en general especificando un objeto o documento en particular).
- El servidor devuelve una respuesta conteniendo el estatus y el contenido de la respuesta (el objeto requerido por el cliente).
- Se termina (corta) la conexión.

En este momento, la especificación validada del HTTP es la 1.0 (la previa fue la 0.9). Aunque el nombre da la idea de que el HTTP sólo es capaz de transmitir hipertextos no es así. Está diseñado con la extensibilidad necesaria para transmitir cualquier tipo de objeto.

Algunas de las características de HTTP son:

- Simplicidad: permite que el servidor maneje poca carga por cada pedido, de manera que puede atender más pedidos simultáneamente.
- Flexibilidad: permite tipificar y transmitir cualquier tipo de dato.
- Sin conexión (*conectionless*): puede haber sólo un pedido por conexión y luego se corta la misma, de manera que la utilización de recursos es la mínima necesaria.
- Sin estado (*stateless*): esto significa que no guarda información sobre transacciones previas. Si bien esto permite agilizar mucho el mecanismo, también es una falacia porque muchas veces es necesario manejar información acerca, por ejemplo, de pasos previos de un usuario y esta información se debe manejar forzando los protocolos.
- Permite manejar metainformación: es información acerca de información. Esto permite que el agente HTML (el browser) decida la mejor manera de aprovechar la misma. Pro ejemplo, podría mandarse un objeto y especificar

el idioma de manera que se puede elegir si se desea hacer el download o no, o la fecha de expiración.

2.3 EJEMPLO DE ARQUITECTURA CLIENTE/SERVIDOR:

2.3.1 Introducción:

Este apartado tiene como objetivo el presentar un ejemplo de arquitectura Cliente/Servidor para conseguir comprender mejor todo lo que se presentará después en capítulos posteriores.

Este ejemplo trata de describir un nuevo protocolo a nivel de aplicación llamado Gnutella, destinado al almacenamiento y búsqueda de información en ambientes distribuidos bajo la modalidad de trabajo persona a persona.

Dado que Gnutella nació como una aplicación de usuario final sobre la que actualmente no hay especificaciones de protocolo, se pretende formalizar su modo de operación y estructuras de datos utilizadas con vías al desarrollo de aplicaciones basadas en esta modalidad cooperativa de trabajo.

Los servicios ofrecidos en Internet, en su mayoría, están implementados bajo el modelo de procesamiento denominado Cliente/Servidor. A diferencia de este esquema altamente centralizado, Gnutella define una red a nivel aplicación, bajo la modalidad persona a persona (*peer to peer*), donde todo nodo (en adelante gnodo) realiza al mismo tiempo funciones de cliente y servidor. Conceptualmente, es un sistema distribuido para almacenamiento y búsqueda de información.

Para comprender las características y el estado de desarrollo del protocolo, resulta necesario remitirse a sus orígenes. Gnutella surgió como un proyecto independiente de los programadores Justin Frankel y Tom Pepper, fundadores de la empresa Nullsoft, destacada por ser la dueña del producto Winamp (reproductor de archivos de sonido mp3).

El día 14 de marzo del año 2000, en el sitio Slashdot (http://www.slashdot.com), se publicó una versión, en fase beta, del programa Gnutella descrito como "una herramienta de software para compartir archivo que puede ser aún más potente que Napster".

Inmediatamente la página fue puesta fuera de servicio, pero ya habían sido descargadas numerosas copias. Nullsoft, ante el hecho en cuestión, se reservó el derecho de publicación (acción que no volvió a ser realizada) aduciendo que la alta demanda colapsó el sitio.

Actualmente, existe una continuación al proyecto llevada a cabo por una comunidad de programadores. Basándose en técnicas de ingeniería inversa y análisis de protocolos han logrado decodificar y realizar varias implementaciones (clones) del protocolo Gnutella.

Es importante destacar dos características fundamentales que hacen a Gnutella merecedor de estudios por parte de grupos de investigación y empresas: su carácter descentralizado y su espíritu cooperativo. A partir de esto, han surgido diversos proyectos basados en el modelo persona a persona en redes amplias.

2.3.2 Sistemas Distribuidos Emergentes para Compartir Archivos:

Un Sistema Distribuido para Compartir Archivos (SDCA) permite que usuarios remotos que operan en máquinas distribuidas puedan intercambiar archivos. Si en este modelo cada máquina puede ofrecer una porción de su sistema de archivos ó solicitar por un archivo remoto (cada máquina es igual en funcionalidad), el sistema es considerado como persona a persona. Esto permite el desarrollo de nuevas aplicaciones y servicios basados en usuarios finales.

El avance de las investigaciones en esta línea permitirá un mejor aprovechamiento de los recursos informáticos. Por ejemplo, una empresa que posea un importante número de computadoras puede combinarlos para potenciar su disponibilidad de recursos (espacio de almacenamiento y tiempo de CPU) y llevar a cabo una determinada tarea.

Ejemplos de modelos de SDCA son los siguientes:

- Freenet: es una red persona a persona diseñada para permitir la distribución de información sobre Internet de manera eficiente. Opera de modo descentralizado y provee un importante grado de anonimato a los usuarios. En Freenet no existen equipos que brinden servicios centralizados y que su caída pueda atentar contra la estabilidad de la red. Utiliza políticas de propagación y de caché más desarrolladas que las de Gnutella, siendo más eficiente y escalable. Implementa estrategias de replicación de archivos frecuentemente accedidos, por lo que permite establecer automáticamente ruteos alternativos hacia réplicas, con el objetivo de regular el tráfico y lograr una transferencia de información más eficiente.
- Napster: es un sistema para compartir archivos mp3 con una base de datos centralizada de archivos y usuarios. Un servidor central es el que vincula las peticiones de los clientes con sus posibles resultados. Una vez que un cliente selecciona un archivo para descargar, basándose en un listado devuelto por el servidor central, el cliente inicia una conexión con la máquina remota que contiene el archivo. El servidor central de Napster nunca tiene archivos almacenados.
- Scour Exchange: es un sistema similar a Napster, salvo por que no solamente se limita a operar con archivos mp3. Permite a usuarios consultar y recuperar información que está en unidades de almacenamiento de usuarios remotos.

2.3.3 Características de Gnutella

Bajo Gnutella se trabaja en un modelo de ambiente distribuido. Un gnodo “X” ofrece abiertamente los archivos que desee compartir con otros usuarios, mientras que a la vez el usuario dueño del gnodo “Y” puede estar recuperando, como cliente, archivos de un gnodo X.

Normalmente las tareas de servidor y cliente están siempre activas en un nodo Gnutella, pero nada impide que alguna pueda no ser activada.

La red Gnutella se compone de numerosos gnodos distribuidos a lo largo del mundo. Su topología no indica jerarquía alguna, dado que cada gnodo es igual a los demás en funcionalidad.

Debido a que uno de los atributos que caracterizan a un gnodo es su ancho de banda ofrecido para descarga de archivos, el modelo sufre ciertas variaciones. Aquellos gnodos de mayor ancho de banda son preferidos a otros, formando hubs ó anillos centrales de conexión a la red.

Cada gnodo de Gnutella sólo sabe acerca de los gnodos con los que se conecta directamente. Todos los otros gnodos son invisibles, a menos que ellos se anuncien contestando a un mensaje tipo ping o contestando a una consulta. Esto proporciona un alto grado de anonimato.

De forma resumida, la red Gnutella opera bajo el modelo conocido como propagación viral. Un cliente envía un mensaje a un gnodo y éste lo reenvía a todos los gnodos a los cuales esté conectado. De esta forma, con sólo conocer la dirección de red de un gnodo, ya se puede ingresar a la misma.

El ingreso a la red se realiza indicando la dirección IP y puerto TCP de cualquier gnodo que ya esté conectado. El gnodo que inicia la conexión anuncia su presencia mediante el envío de un mensaje. El gnodo al cual se conectó reenvía este mensaje a todos los gnodos a los cuales esté conectado directamente y así sucesivamente. Cada uno de estos gnodos, responde a este mensaje indicando cuántos archivos comparte y tamaño total de los mismos.

Un ejemplo del alto grado de estabilidad e independencia que presenta el modelo puede verse en el siguiente ejemplo: suponga que un gnodo-1 se conecta un gnodo-2; todo lo que ofrezca gnodo-1 puede ser tomado por gnodo-2 y viceversa.

Ahora gnodo-3 se conecta a gnodo-2 y éste le informa a qué gnodos está directamente conectado (los mensajes de gnodo-3 pueden llegar a gnodo-1 dado que los reenviará gnodo-2). A continuación gnodo-2 se pone en estado fuera de servicio.

La red sigue funcionando normalmente dado que gnodo-3 posee direcciones alternativas de gnodos pasadas anteriormente por gnodo-2, que le permitirán conectarse a gnodos alternativos para seguir vinculado a la red.

En las implementaciones del protocolo Gnutella, un parámetro a definir es la cantidad de conexiones simultaneas a la red que siempre un gnodo debe tratar de mantener. Cuantas más conexiones tenga, se logrará un mayor espacio de consulta y se asegurará una alta disponibilidad de la red.

2.3.4 Estructura de Mensajes:

Un mensaje gnutella se transporta sobre protocolo TCP. Su cabecera siempre consta de 23 bytes y está formado por los siguientes campos:

- Identificador de gnodo (16 bytes).
- Función (1 byte).

- TTL o tiempo de vida restante (1 byte).
- Hops o saltos (1 byte).
- Largo de la carga en bytes (4 bytes).

El valor referenciado en el campo función indica qué tipo de acciones deben realizar aquellos gnodos que reciben el mensaje. La cabecera descrita es fija para cualquier tipo de función instanciada; lo que es variable en tamaño es la segunda parte o carga del mensaje dado que depende directamente de la función.

Gnutella para operar necesita cinco funciones que le permitirán mantenerse en la red, consultar por recursos y descargar archivos.

- **Función Ping o Init:** Se utiliza para que los gnodos anuncien su presencia en la red. No lleva carga de mensaje, es decir, que el campo largo de la carga se instancia en cero. Cada gnodo que reciba un mensaje Ping debe responder generando mensajes Pong (pueden ser n) donde anuncien su presencia o de gnodos que tengan noticia. Un gnodo generará un Ping, además, para recabar información sobre gnodos vecinos.

  Ejemplo mensaje Ping:

  27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62 00 07 00 00 00 00 00

  Mensaje:

  - Identificador de gnodo: 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
  - Función: 00 TTL: 07
  - Hops: 00
  - Largo de la carga: 00 00 00 00

- **Función Pong ó Init_Response:** Un gnodo envía un mensaje Pong como respuesta a un mensaje Ping. La carga de un Pong se compone de los siguientes campos:

  - Puerto (2 bytes).
  - Dirección IP (4 bytes).
  - Cantidad de archivos compartidos (4 bytes).
  - Kbytes compartidos (4 bytes).

  Es interesante destacar que un gnodo, al recibir datos como los descritos, puede abrir nuevas conexiones que le permitan extender su dominio de acción y, a la vez, darle una mayor estabilidad a su presencia en la red Gnutella. Un gnodo puede enviar más de un mensaje Pong como respuesta a un único Ping, dado que informa acerca de otros gnodos que tenga almacenado en su memoria caché.

Ejemplo mensaje Pong:

27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62 01 07 00 0E 00 00 00 CA 18 AA D2 62 95 03 00 00 00 00 00 00 00

Mensaje:

- Identificador de gnodo: 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
- Función: 01
- TTL: 07
- Hops: 00
- Largo de la carga: 0E 00 00 00

Carga:

- Puerto: CA 18
- Dirección IP: AA D2 62 85
- Cantidad de archivos compartidos: 03 00 00 00
- Kbytes compartidos: 00 00 00 00

- **Función Push:** Es utilizada por gnodos que acceden a la red atravesando un firewall y no pueden descargar un recurso situado en un gnodo al otro lado del mismo. El gnodo que requiere el recurso enviará un mensaje Push al gnodo que lo tiene con los siguientes campos:

  - Identificador de gnodo (16 bytes).
  - Índice (4 bytes).
  - Dirección IP (4 bytes).
  - Puerto (2 bytes).

  Una vez obtenido el mensaje, el gnodo poseedor del recurso establecerá una conexión con el gnodo peticionante y le enviará a la dirección IP y puerto informado el archivo referenciado por el campo índice. Una respuesta a la función Push no está definida, dado que un gnodo puede enviar el recurso o no hacer caso de la petición.

Ejemplo mensaje Push:

27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62 40 07 00 1A 00 00 00 22 11 3F 71 49 B2 D4 11 88 23 00 80 AD 40 22 11 02 00 00 00 AA 12 62 AA CA 22

Mensaje:

- Identificador de gnodo: 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
- Función: 40
- TTL: 07
- Hops: 00
- Largo de la carga: 1A 00 00 00

Carga:

- Identificador de gnodo: 22 11 3F 71 49 B2 D4 11 88 23 00 80 AD 40 22 11
- Índice: 02 00 00 00
- Dirección IP: AA 12 62 AA
- Puerto: CA 22

- **Función Query:** Se utiliza para que un gnodo pueda consultar a otros gnodos por un recurso. La carga utilizada por esta función se compone de dos campos, a saber:

  - Velocidad mínima requerida para descarga (2 bytes).
  - Criterio de búsqueda de longitud variable.

  Un gnodo al recibir un mensaje Query y constatar que no hay coincidencias en su estructura de archivos no generará ningún mensaje de respuesta.

Ejemplo mensaje Query:

28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62 80 07 00 06 00 00 00 00 00 2A 2E 61 00

Mensaje:

- Identificador de gnodo: 28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
- Función: 80
- TTL: 07
- Hops: 00
- Largo de la carga: 06 00 00 00

Carga:

- Velocidad mínima: 00 00
- Criterio de búsqueda: 2A 2E 61 00 (*.a + carácter nulo)

- **Función Query_Hit:** Se utiliza como respuesta a un mensaje Query. Los campos que componen un Query_Hit son:

  - Cantidad de hits o coincidencias (1 byte).
  - Puerto (2 bytes).
  - Dirección IP de descarga (4 bytes).
  - Velocidad de operación (4 bytes).
  - Resultados (longitud variable).
  - Identificador del gnodo que responde (16 bytes).

  Los resultados (que pueden ser n) se estructuran como una lista de la siguiente forma:

  - Índice o número de respuesta (4 bytes).
  - Tamaño del archivo (en Kbytes) (4 bytes).
  - Nombre del archivo (longitud variable).
  - Delimitador de registro (0x0000).

  El gnodo que responde incluye en la respuesta su identificador para que lo pueda utilizar el gnodo peticionario, si decidiera solicitar un recurso mediante un mensaje Push.

Ejemplo mensaje Query_Hit:

28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62 81 07 00 57 00 00 00 03 CA 18 AA D2 62 95 1C 00 00 00 00 00 00 00 70 00 00 00 31 34 39 61 72 63 68 69 2E 61 00 00 01 00 00 00 70 00 00 00 31 34 39 61 72 63 68 69 2E 62 00 00 02 00 00 00 70 00 00 00 31 34 39 61 72 63 68 69 2E 6D 00 00 A0 3D 8D 3D 7D 84 D1 11 85 8E 52 54 00 DC 37 66

Mensaje:

- Identificador de gnodo: 28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
- Función: 81
- TTL: 07
- Hops: 00
- Largo de la carga: 57 00 00 00

Carga:

- Cantidad de hits: 03
- Puerto: CA 18
- Dirección IP: AA D2 62 95
- Velocidad: 1C 00 00 00

Estructura de resultados:

- 1er elemento:

  Índice: 00 00 00 00
  Tamaño de archivo: 70 00 00 00
  Nombre de archivo: 31 34 39 61 72 63 68 69 2E 61 (149archi.a)
  Terminador: 00 00

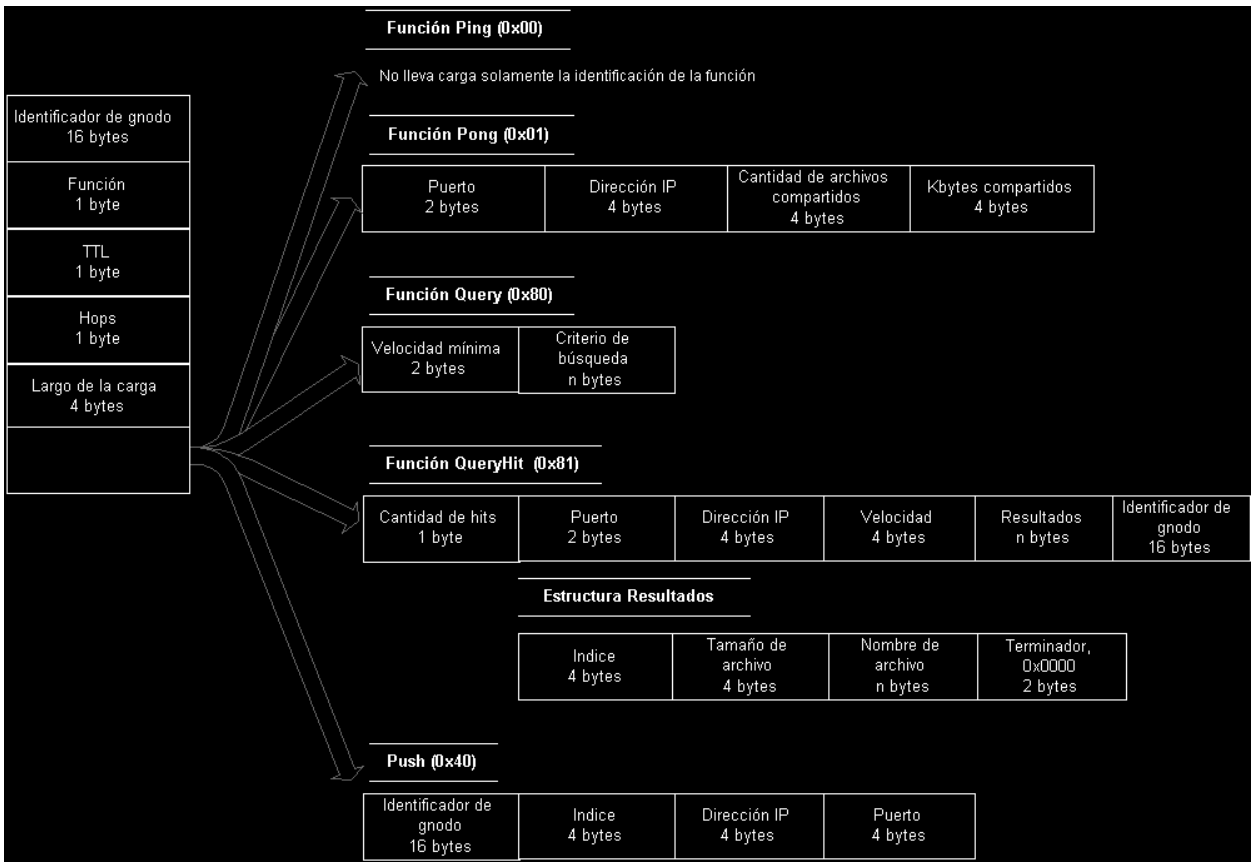
- 2do elemento:

  Índice: 01 00 00 00
  Tamaño de archivo: 70 00 00 00
  Nombre de archivo: 31 34 39 61 72 63 68 69 2E 62 (149archi.b)
  Terminador: 00 00

- 3er elemento:

  Índice: 02 00 00 00
  Tamaño de archivo: 70 00 00 00
  Nombre de archivo: 31 34 39 61 72 63 68 69 2E 6D (149archi.m)
  Terminador: 00 00

Con el siguiente esquema se trata de representar la estructura de datos de mensajes Gnutella.

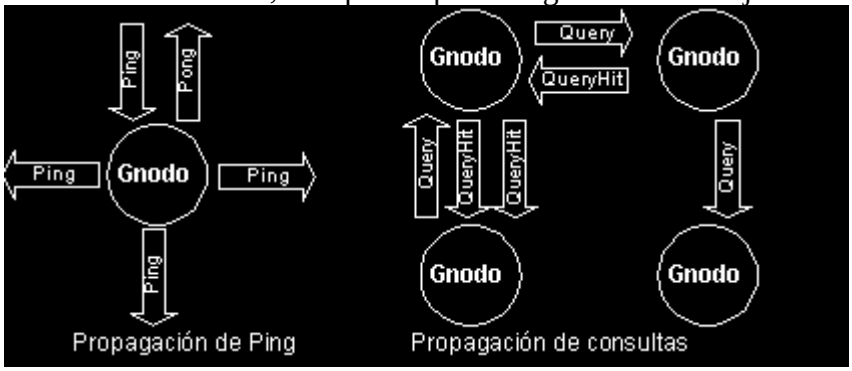


2.3.5 Reglas de Propagación:

El modelo persona a persona implantado en Gnutella requiere que los gnodos tengan la capacidad de propagar los mensajes recibidos (funciones Ping, Pong, Query, Query_Hit y Push) a través de sus conexiones establecidas.

A los efectos de hacer eficiente la operación de la red (teniendo en cuenta su topología, la ausencia de jerarquía y la existencia de loops), deben implementarse en cada gnodo una serie de reglas de propagación que a continuación se detallan:

- Un gnodo propagará mensajes tipo Ping y Query a todos sus gnodos conectados directamente, excepto al que entregó dicho mensaje.



- Los mensajes tipo Pong, Query_Hit y Push sólo deben ser propagados por el mismo camino por el que viajó el mensaje inicial (Ping o Query) asociado. Esto es posible debido a que, antes de propagar un mensaje, los gnodos revisan en tablas internas por aquel que generó tal respuesta y así obtienen la conexión por donde deben reenviarlos.
- Un gnodo decrementará en 1 el valor del campo TTL de un mensaje e incrementará el valor del campo Hops en 1 antes de propagar dicho mensaje por las conexiones pertinentes. Si al decrementar el valor de TTL el resultado obtenido es cero, debe desechar el mensaje.
- Si un gnodo recibe un mensaje con idéntico identificador de gnodo y mismo valor de función que uno recibido anteriormente (en un período breve de tiempo, no especificado formalmente), debería evitar propagarlo.

En un instante de tiempo puede haber miles de gnodos en la red, pero para un gnodo en particular su dominio de acción estará restringido a la cantidad de conexiones establecidas y al alcance de sus mensajes (antes que el valor del campo TTL llegue a cero y sea descartado).

Debido al dinamismo, la topología y la existencia de usuarios temporales, es posible que las respuestas a una misma consulta varíen en el tiempo. Cada vez que un usuario ingresa en la red puede hacerlo a una parte diferente de la misma, según el estado actual de sus gnodos vecinos.

2.3.6 Descarga de archivos:

Una vez que un gnodo recibe un resultado (Query_Hit) a una consulta previamente realizada, puede optar por seleccionar un archivo para su descarga desde el gnodo remoto. Los archivos se recuperan directamente, sin intervención de gnodos intermedios y, por lo tanto, tampoco de la red Gnutella. La descarga se realiza mediante el uso del protocolo HTTP versión 1.0.

Un gnodo al recibir un Query_Hit obtiene los siguientes datos: dirección IP y puerto donde el gnodo que posee el recurso está en modo de apertura pasiva, índice, tamaño y nombre del archivo.

A los efectos de iniciar la descarga, el gnodo que requiere el recurso abrirá una conexión TCP contra el gnodo remoto en el puerto anteriormente obtenido y, mediante la primitiva GET del protocolo HTTP, solicitará el recurso seleccionado indicando índice y nombre de archivo.

Ejemplo:

GET /// HTTP/1.0 Connection: Keep-Alive Range: bytes=0- e.

El gnodo remoto, al recibir tal petición y validar que el recurso solicitado está disponible, transmitirá el archivo previo envío de una cabecera estándar de HTTP con el siguiente formato:
HTTP 200 OK
Server: Gnutella
Content-type: application/binary
Content-length: <tamaño del archivo en bytes>

2.3.7 Funcionamiento de la red Gnutella:

A los efectos de describir el comportamiento de una red Gnutella, se realizó una experiencia de laboratorio de la cual se obtuvo una muestra para el análisis del protocolo. Se utilizaron cuatro equipos corriendo el software Gnutella versión 0.56, con lo que cada uno de éstos actuó como gnodo. A continuación se describe cada uno:

Dirección IP del gnodo: 170.210.98.2 (en gráfico se hace referencia al último byte)
Identificador de gnodo: 4E62 (cuatro últimos bytes)
Archivos compartidos: 2archi.a, 2archi.b y 2archi.z

Dirección IP del gnodo: 170.210.98.3
Identificador de gnodo: XXXX
Archivos compartidos: 3archi.a, 3archi.b y 3archi.i

Dirección IP del gnodo: 170.210.98.150
Identificador de gnodo: 4D60
Archivos compartidos: 150archi.a, 150archi.x y 150archi.p

Dirección IP del gnodo: 170.210.98.149
Identificador de gnodo: 3760
Archivos compartidos: 149archi.a, 149archi.b y 149archi.m

En los gráficos, cada flecha indica el sentido del mensaje y la identificación de la misma consta de cuatro elementos:

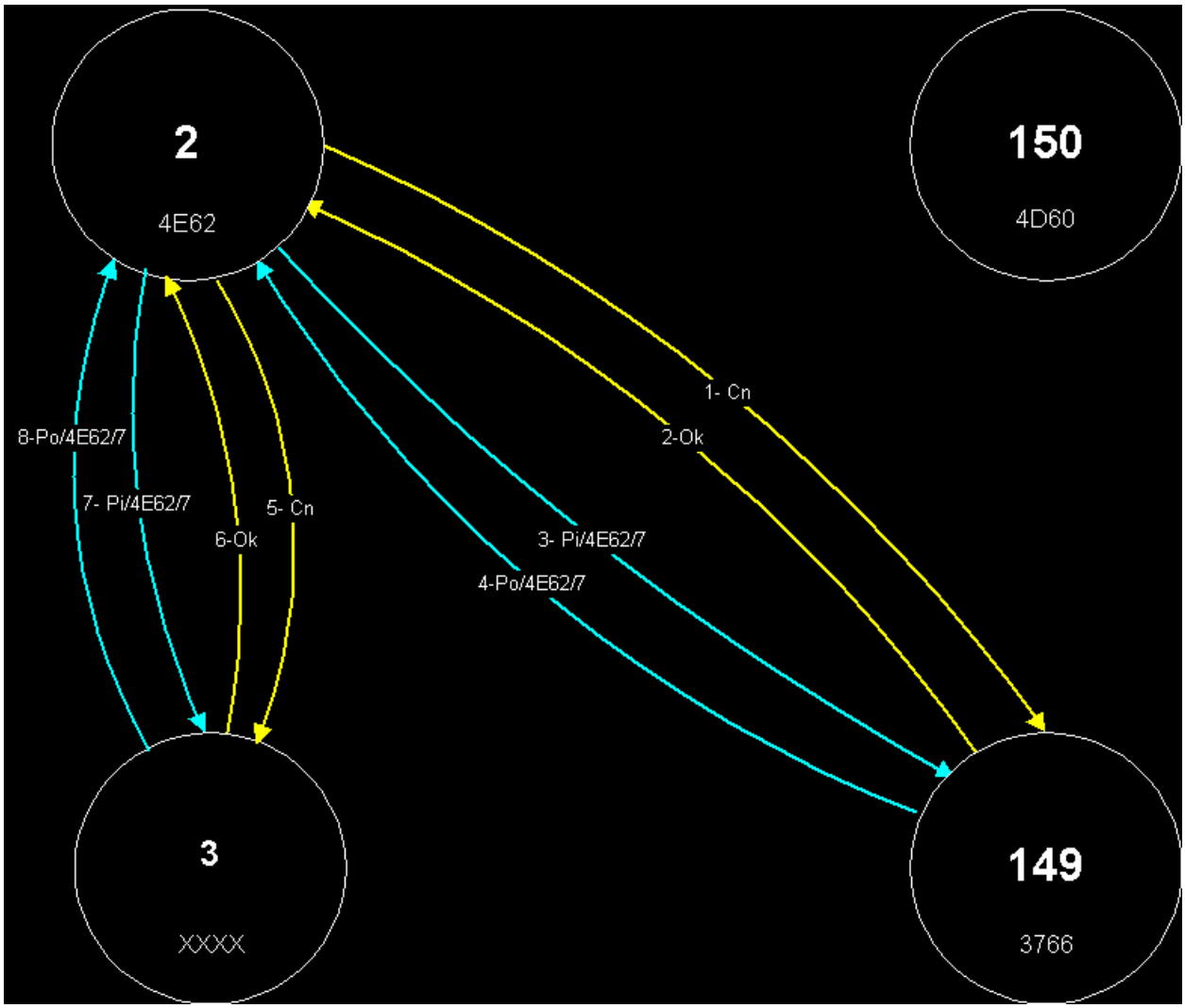
- Número de mensaje.
- Tipo de mensaje.
- Identificación del origen.
- TTL.

La flecha gruesa indica una conexión abierta y su sentido (desde el punto de vista de la apertura). Los tipos de mensaje están codificados de la siguiente manera:

Cn: GNUTELLA CONNECT - Apertura de conexión a nivel aplicación
Ok: GNUTELLA OK - Aceptación de apertura de conexión
Pi: PING - Ping
Po: PONG - Pong
Qy: QUERY - Consulta
Rs: QUERY_HIT - Respuesta

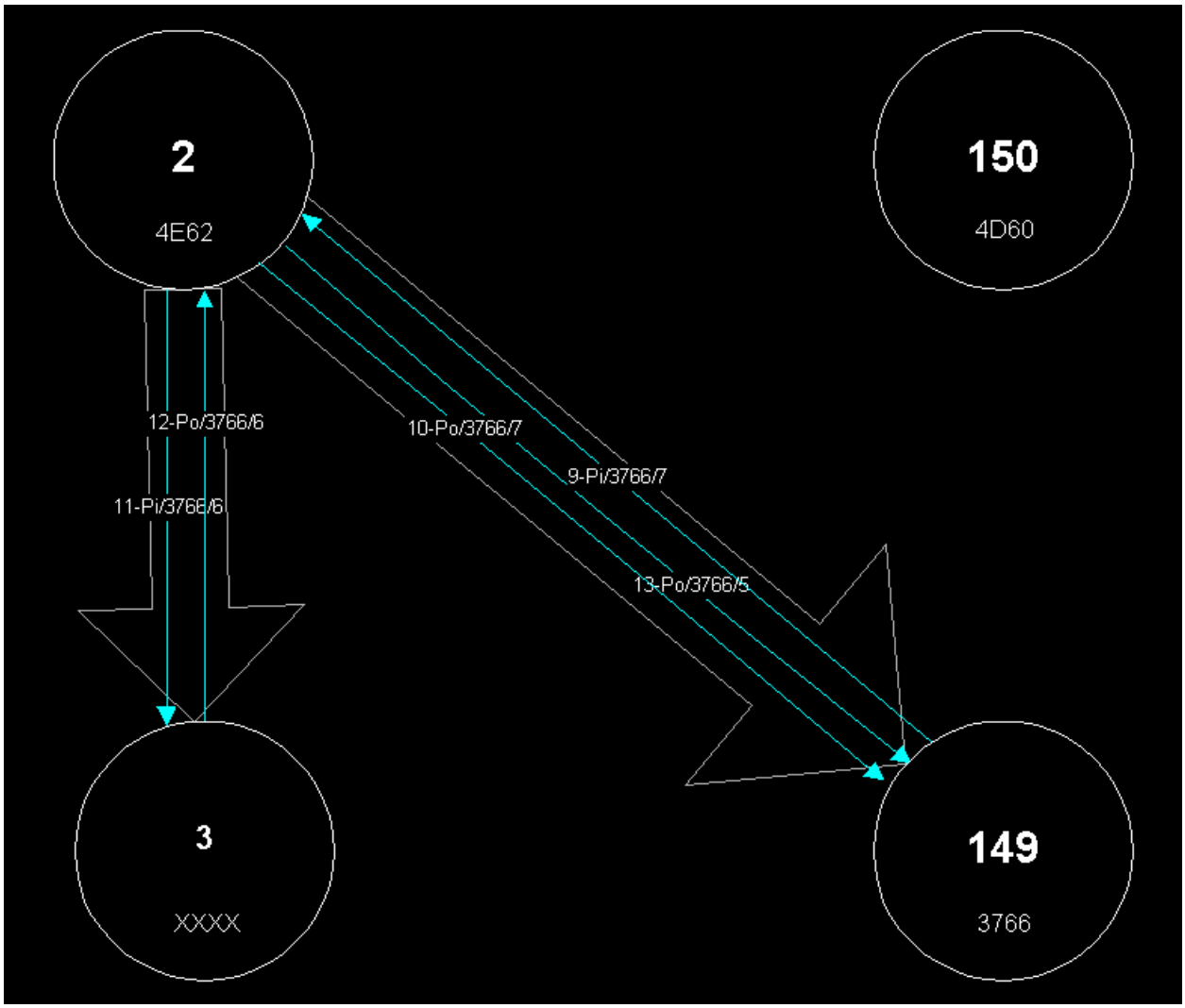
La secuencia de operaciones fue la siguiente:

El primer lugar, se solicitó al gnodo 2 la apertura de comunicación con el gnodo 149 y luego con el gnodo 3. Una vez aceptada la apertura de conexión a nivel aplicación, el gnodo 2 envió un mensaje Ping a cada uno de ellos, obteniendo un mensaje Pong como respuesta.

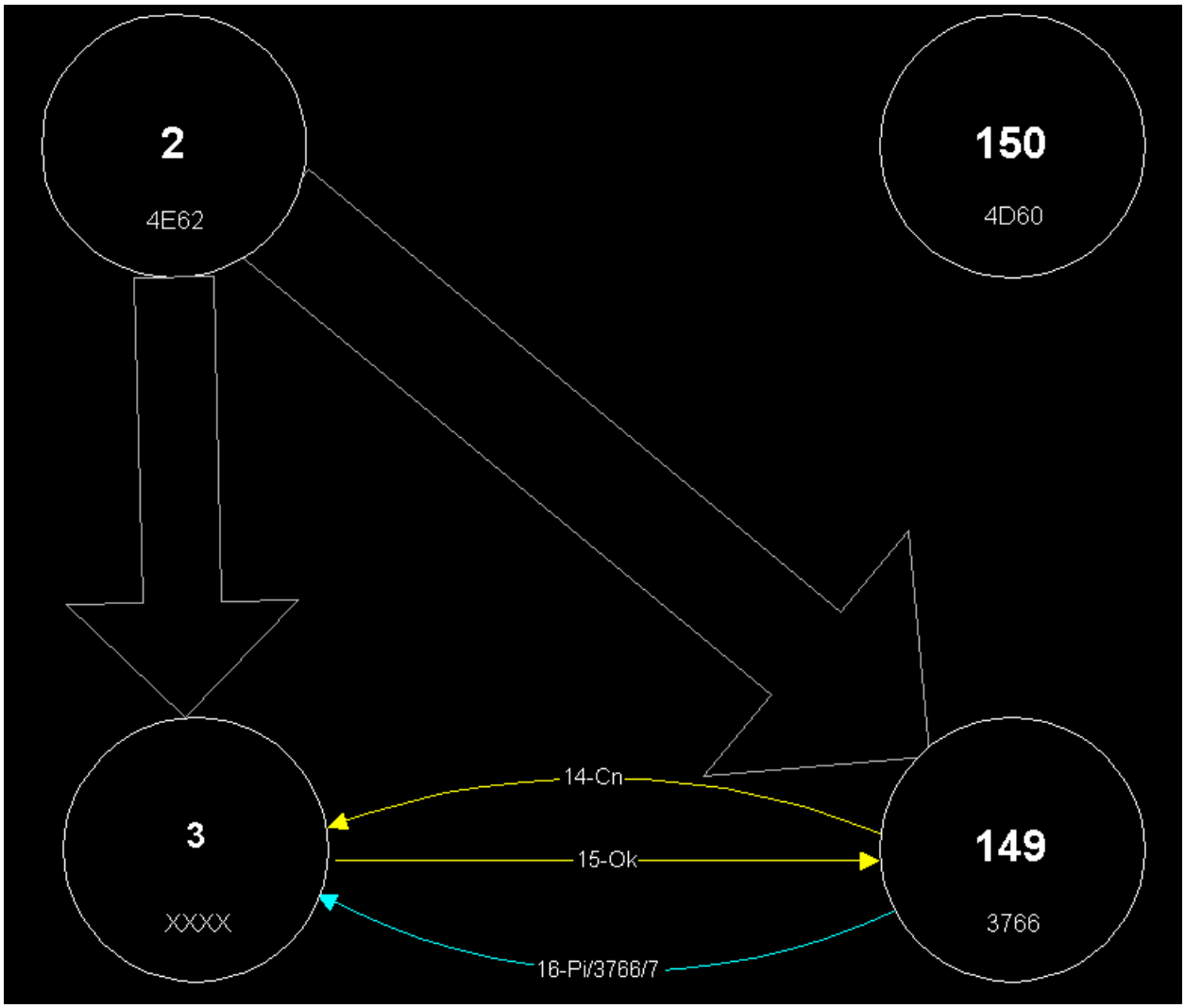


A continuación, se solicitó al gnodo 149 la ejecución de un comando de aplicación de actualización de la información de conexiones Gnutella y recursos compartidos, lo que generó el envío de un mensaje Ping por todas sus conexiones abiertas. Como sólo tenía conexión con el gnodo 2, le envió el Ping a éste, quien primero lo contestó (con su mensaje Pong) y luego lo propagó por todas sus conexiones abiertas, excepto por donde lo recibió (en este caso con el gnodo 3) decrementando el TTL del mensaje.

Gnodo 3 contestó al gnodo 2 al Ping y éste lo redireccionó al gnodo 149, que fue quien originó el requerimiento.

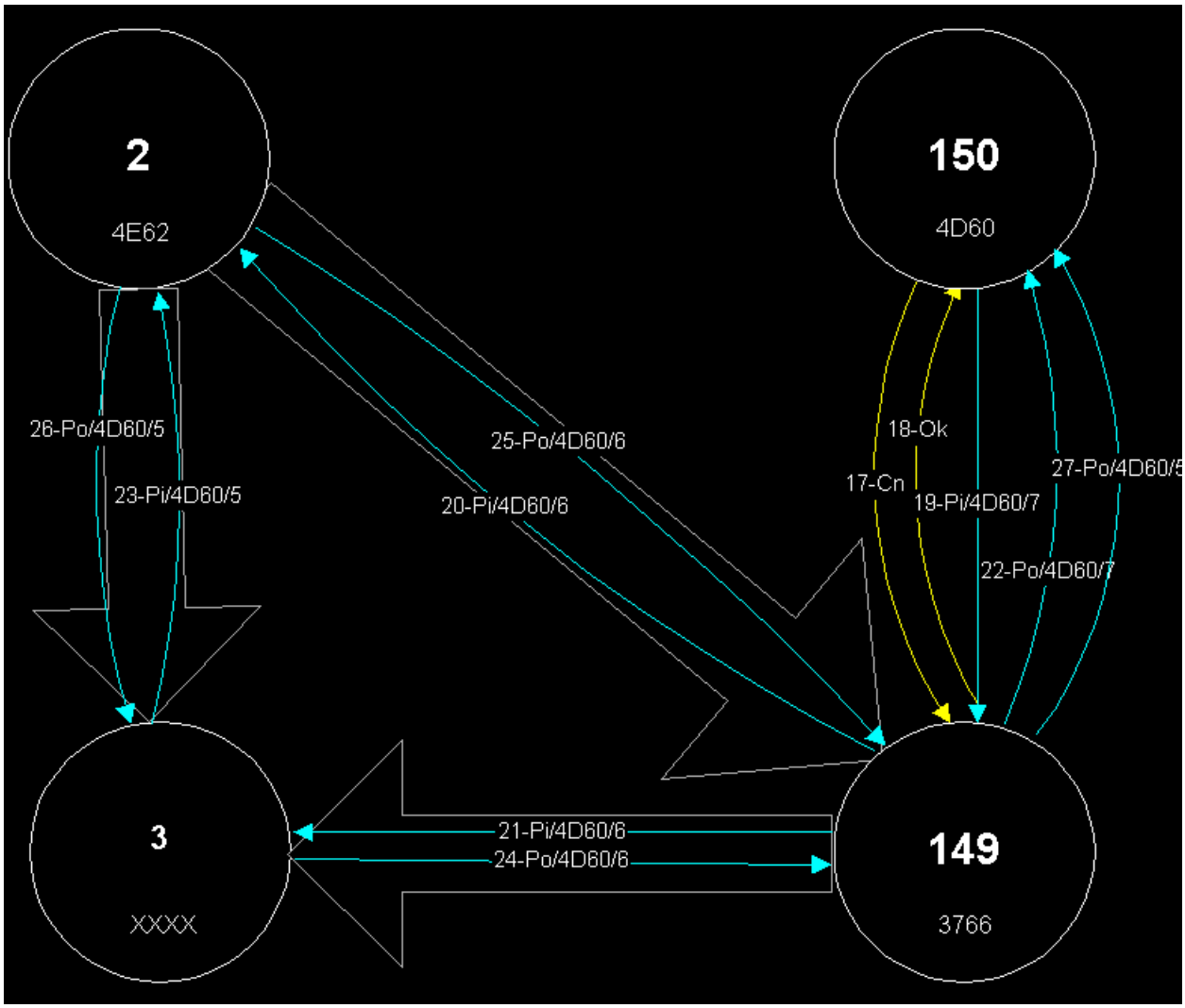


Al recibir información de la existencia del gnodo 3, el gnodo 149 abrió una conexión con éste y le envió un mensaje Ping.

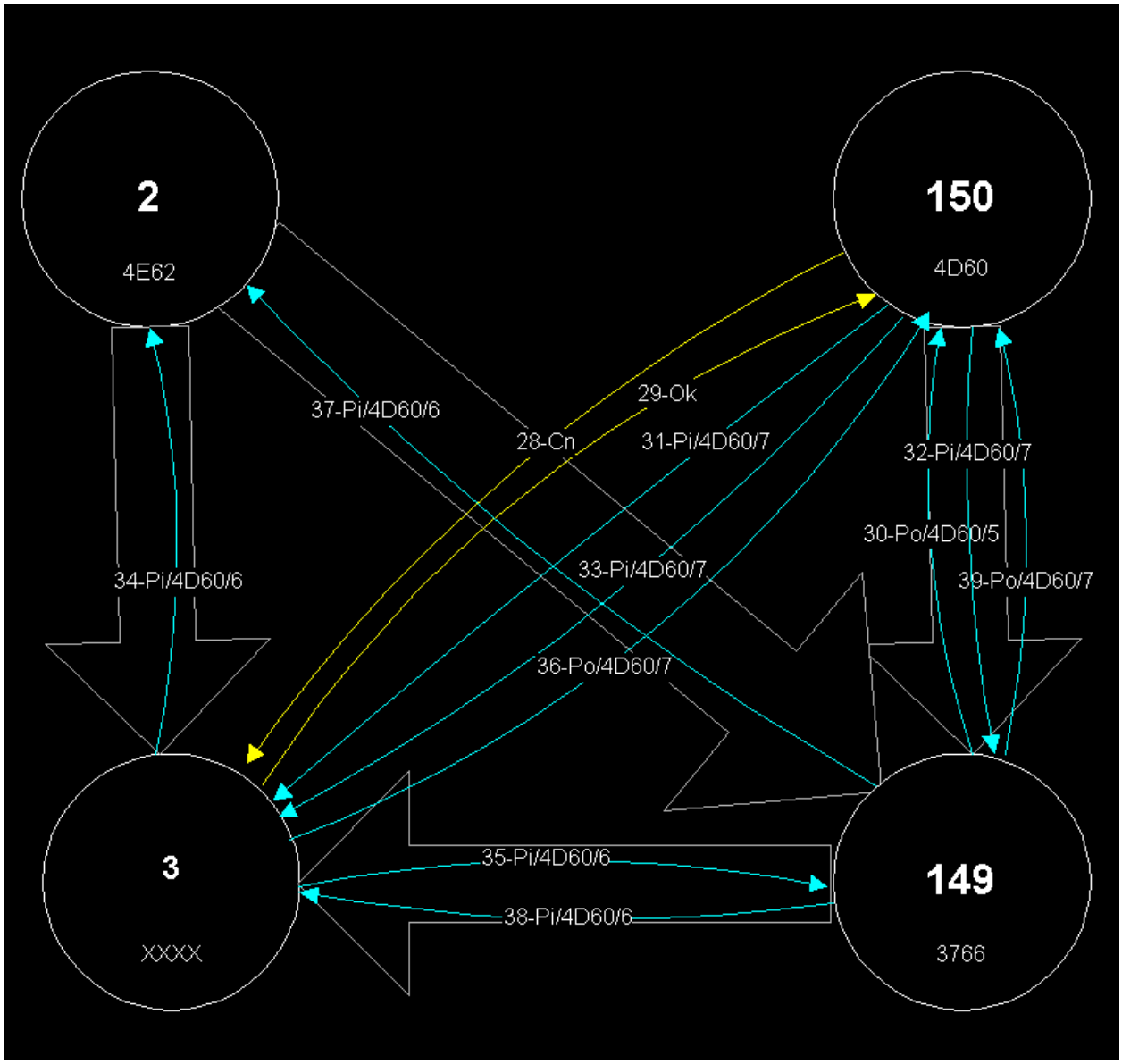


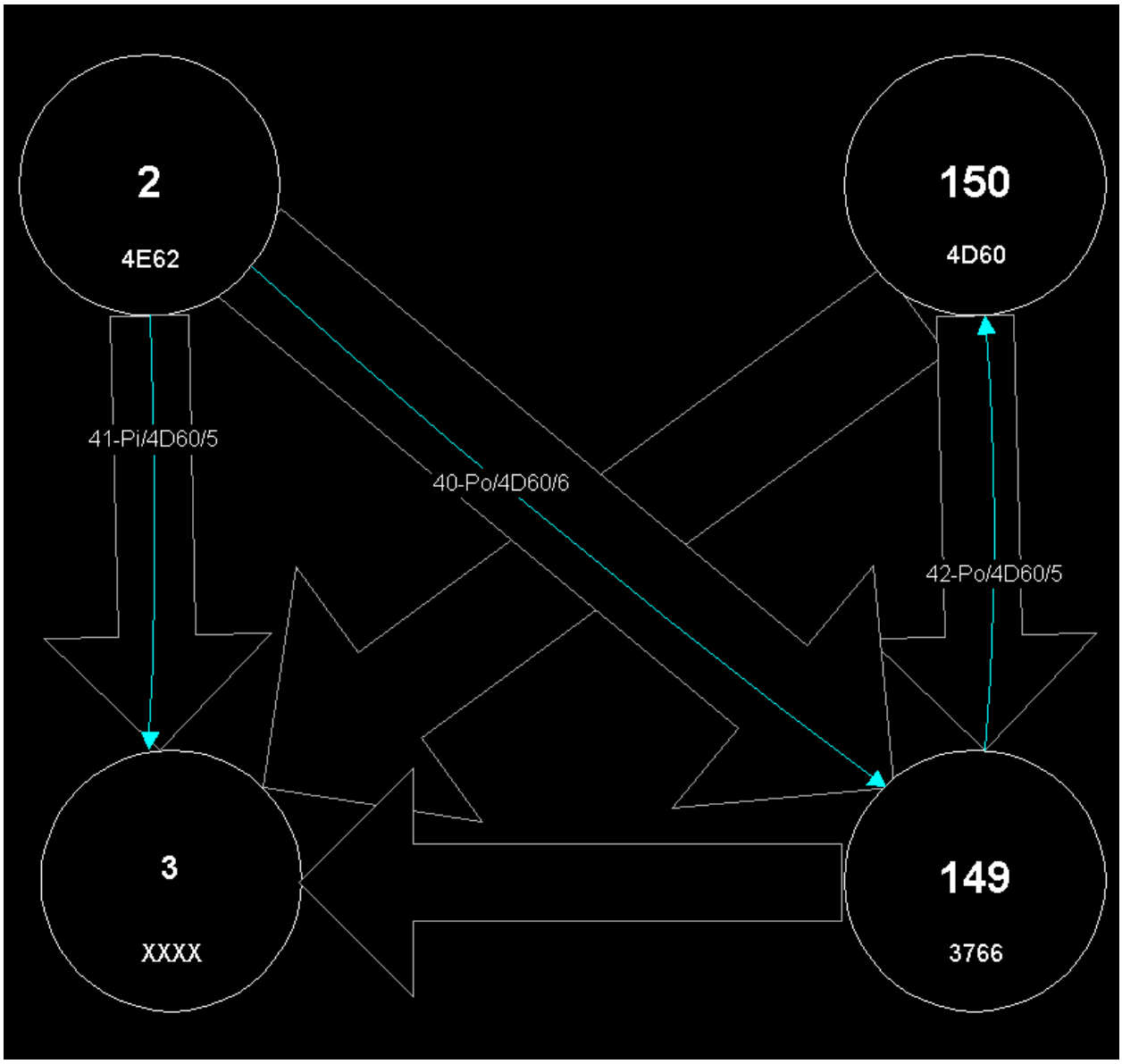
Luego, se solicitó al gnodo 150 la apertura de una conexión con el gnodo 149. Finalizada la misma, el gnodo 150 envió un mensaje Ping al gnodo 149, el cual lo contestó (Pong) y luego lo propagó por todas sus conexiones abiertas (con los gnodos 2 y 3, quienes a su vez hicieron lo propio).

Las respuestas viajaron por los mismos caminos por donde circuló el mensaje Ping y, finalmente, llegaron al gnodo 150 (con el TTL del mensaje decrementado según los gnodos que se atravesaron en el camino).



Al recibir información de la existencia del gnodo 3, el gnodo 150 abrió una conexión con éste y le envió un mensaje Ping, el cual fue propagado según el mismo comportamiento anterior.


2
4E62
150
4D60
3
XXXX
149
3766
37-Pi/4D60/6
29-Ok
28-Cn
31-Pi/4D60/7
32-Pi/4D60/7
30-Po/4D60/5
34-Pi/4D60/6
33-Pi/4D60/7
39-Po/4D60/7
36-Po/4D60/7
35-Pi/4D60/6
38-Pi/4D60/6



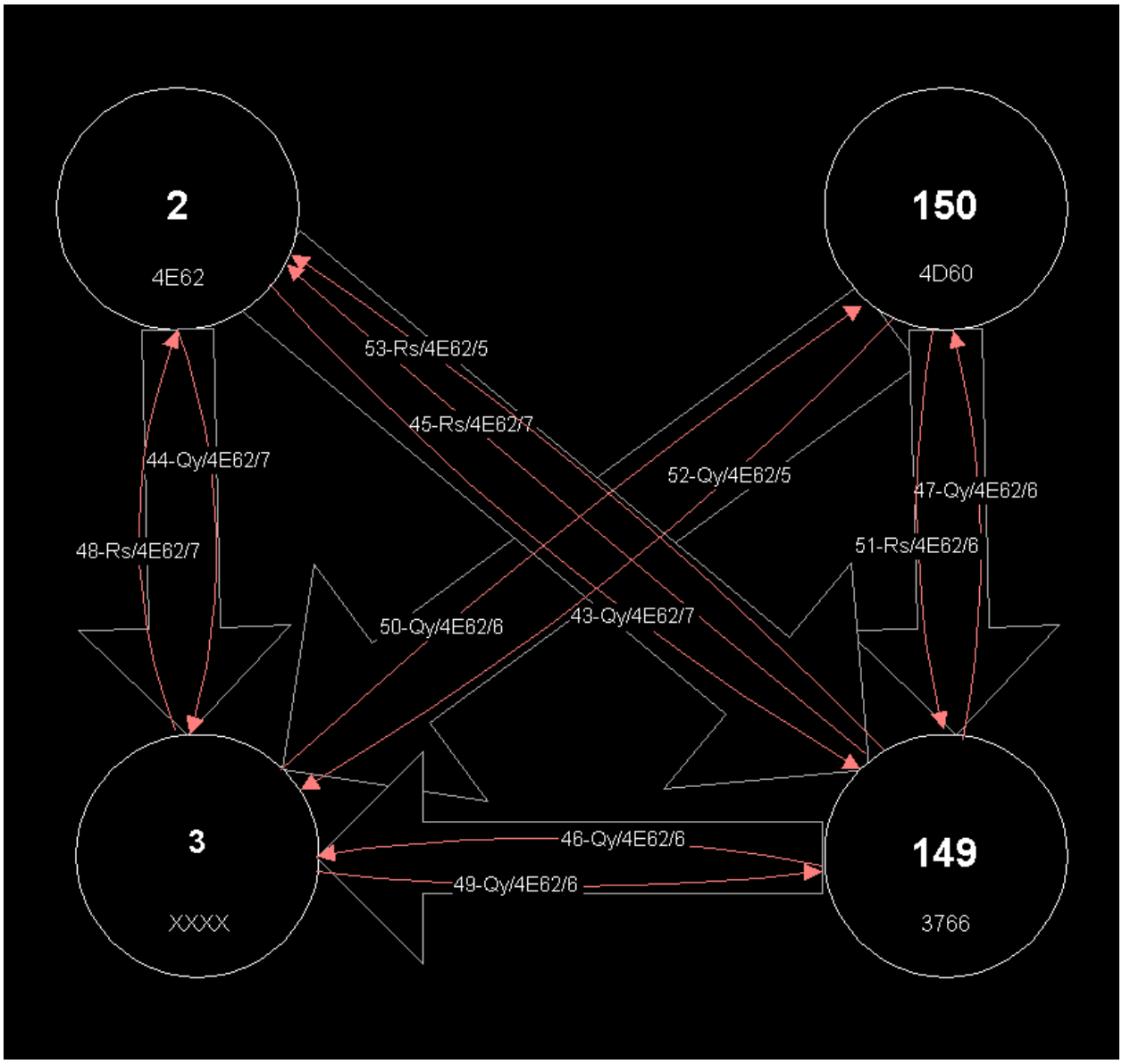
En este momento, la red Gnutella constaba de cuatro gnodos, sobre la cual se ejecutaron las siguientes consultas:

En el gnodo 2 se solicitó la consulta por el patrón de caracteres "*.a". Este gnodo envió la consulta por todas sus conexiones abiertas (en este caso con los gnodos 149 y 3).

Los gnodos 149 y 3, a su vez, propagaron la consulta a través de todas sus conexiones (excepto por donde entró).

Aquellos gnodos que poseían archivos cuyo nombre correspondía al patrón de búsqueda (en este caso los gnodos 2, 149 y 150) respondieron a la consulta enviando un mensaje por el mismo camino de la misma.

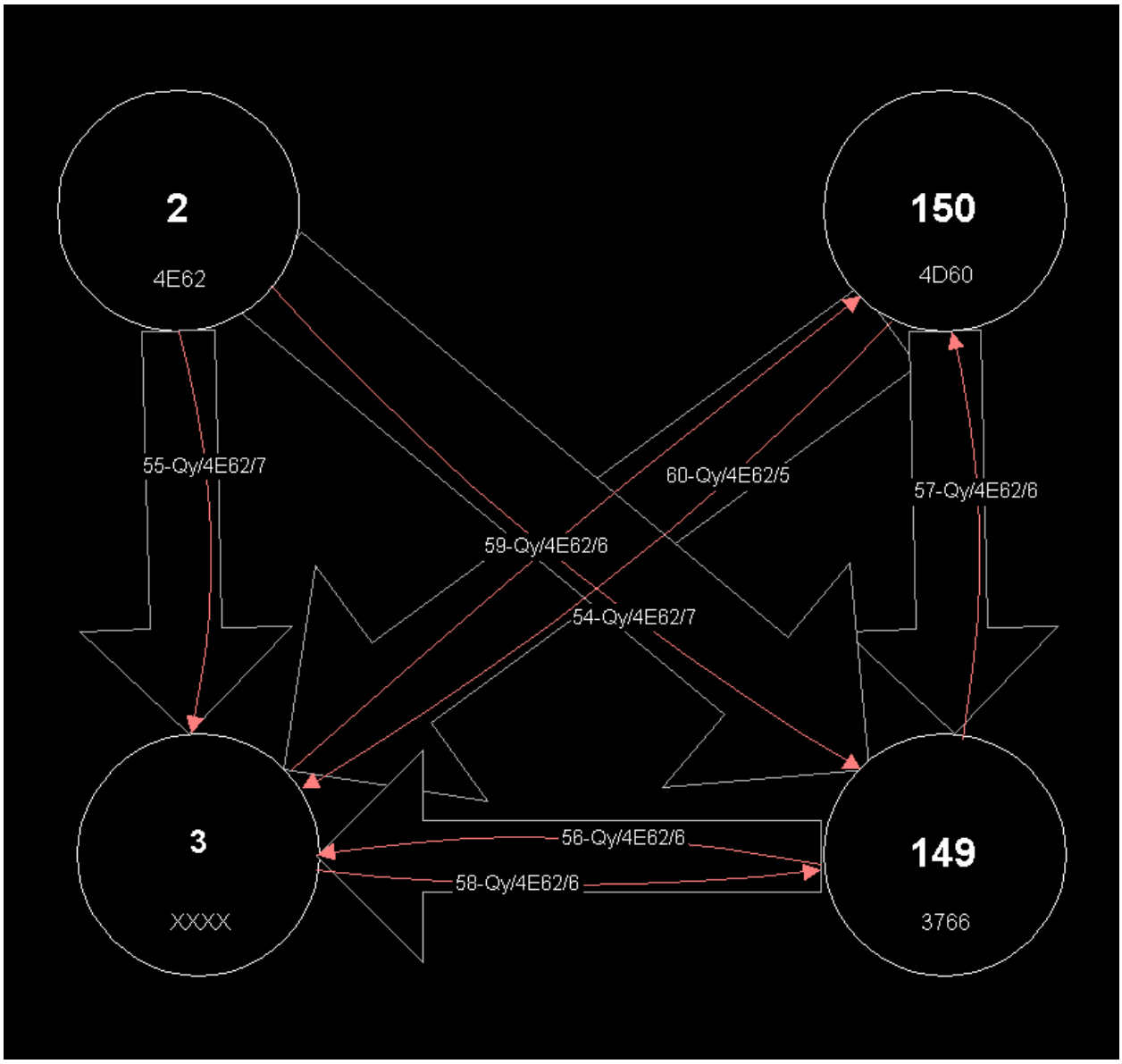
Hay que fijarse en que el comportamiento de los gnodos en cuanto a la propagación de una consulta es similar a la de un mensaje Ping, ocurriendo lo mismo con las respuestas (respecto de los mensajes Pong).



Finalmente, en el gnodo 2 se solicitó la consulta por el patrón de caracteres "i.a".

Este gnodo envió la consulta de la misma forma que la anterior. Los gnodos 149 y 3 propagaron la misma.

Ninguno de los gnodos poseía archivos cuyo nombre correspondía al patrón de búsqueda; por lo tanto, no la respondieron.



2.3.8 Conclusión:

El uso masivo del protocolo en implementaciones sobre diferentes plataformas ha demostrado que Gnutella es un punto de inicio válido para el desarrollo de servicios distribuidos basados en la filosofía persona a persona.

Debido a su génesis y corta existencia, Gnutella es aún un protocolo inmaduro sometido a estudios para evaluar mejoras en eficiencia y funcionalidad (para implementar en versiones posteriores).

Como característica interesante, presenta la definición de una red a nivel aplicación, orientada al cooperativismo, que brinda elementos para favorecer la heterogeneidad, la estabilidad, la redundancia y la tolerancia a fallos.

El modelo de propagación puede servir para generar aplicaciones distribuidas que no solamente se limiten a tareas de compartir archivos.

Por ejemplo, se podría utilizar este modelo, con modificaciones menores, con el objetivo de lograr una red de procesamiento distribuido basada en el cooperativismo.

2.4 DISCUSIÓN:

En este capítulo se han comentado las distintas arquitecturas existentes para la comunicación de un sistema distribuido. Del mismo modo, se han expresado las diferentes elecciones existentes para la comunicación entre los clientes y servidores.

Una buena elección del sistema a utilizar será un punto clave para el desarrollo de cualquier proyecto en el que se vea implicado un sistema que cumpla el paradigma del Cliente/Servidor.

Con todo la información recogida en el presente capítulo, se dispone de la base teórica necesaria para comprender lo que a continuación se expresará en capítulos venideros y que son los que reflejan el marco del proyecto realizado.

Capítulo 3

Solución Propuesta

ÍNDICE DEL CAPÍTULO 3:

3. SOLUCIÓN PROPUESTA:

3.1 Introducción:

En este capítulo se describe el diseño de la plataforma Cliente/Servidor propuesta para la realización del presente proyecto.

En primer lugar, se expone una introducción al proyecto COSM y todo lo que ello conlleva, pasando posteriormente a comentar todo lo relacionado con la plataforma Cliente/Servidor, así como su implementación y las peculiaridades que la hacen tan interesante.

A través de los diferentes apartados, se irá explicando paso a paso el desarrollo de la plataforma y las decisiones que se van tomando al respecto.

Más adelante comentaremos cosas referentes a las pequeñas aplicaciones realizadas para la prueba de la propia plataforma Cliente/Servidor, así como los resultados obtenidos de estas pruebas llevadas a cabo.

Por último, para finalizar el capítulo, se discutirá acerca de la solución tomada, las posibles alternativas existentes y los posibles cambios y mejoras en el futuro.

3.2 El proyecto COSM:

Este proyecto ha sido llevado a cabo en sus inicios por una empresa americana cuyo nombre es Mithral Comunications & Desing INC.

Esta empresa originalmente fue fundada en 1995. La compañía, en la actualidad, está inmersa en llegar a alcanzar el foco central que le permita integrar cualquier tipo de plataforma.

El proyecto principal de Mithral es el COSM, el cual empezó en 1995 y es el centro de este proyecto.

Más que un simple producto, COSM es un proyecto que intenta agrupar e integrar todo tipo de plataformas y potenciar el desarrollo y la simplicidad que todo desarrollador desea.

COSM contiene una completa estructura que le permite soportar cualquier tipo de plataforma o sistema operativo, aunque es mucho más que todo eso.

Esta plataforma se puede correr bajo cualquier tipo de sistema operativo existente en la actualidad, pero en el presente proyecto tan sólo nos hemos ceñido en dos, LINUX y WINDOWS NT, de cuya generación de archivos ejecutables nos ocuparemos más adelante en el presente capítulo.

Toda la plataforma está desarrollada íntegramente en ANSI C y es totalmente compatible con C, C++, aplicaciones realizadas en FORTRAN, etc.

3.2.1 El proyecto COSM. PHASE I:

Con la Fase I de COSM se ha conseguido realizar la plataforma Cliente/Servidor que pretendíamos para el presente proyecto.

Esta fase fue publicada en 1999 y contenía un conjunto de protocolos abiertos y de aplicaciones diseñadas que permitieran a cualquier tipo de computadora que se encontrara en cualquier parte del mundo trabajar conjuntamente con otras en algún proyecto en el que se pretendiera distribuir la carga de trabajo.

Todos esto proyectos para distribuir la carga de trabajo se centran en problemas como retos matemáticos, el rendering en animaciones, escritura, etc.

COSM nos permite diseñar y desarrollar rápidamente una plataforma Cliente/Servidor, la cual incluye la distribución del trabajo en diferentes ordenadores.

Existen algunos ejemplos reales de empresas que han utilizado este proyecto para montar su propia plataforma Cliente/Servidor. Podemos mencionar a:

- distributed.net
- SETI@Home.
- Napster.
- Gnutella.

Actualmente, los dos proyectos más importantes en los que se está haciendo uso de COSM son los siguientes:

- **Folding@Home** – Protein folding: está siendo realizado por el grupo Panda, en su laboratorio de Químicas en la Universidad de Standford. Las proteínas son la base para que la biología pueda explicar infinidad de cosas. De este modo se ha sacado, o mejor dicho, se ha comprendido y descifrado el genoma humano, es decir, la guía de todas las proteínas en biología. Un paso importante es el conocer cómo se ensamblan las diferentes proteínas entre sí. Esto requiere un extremado nivel computacional, ya que la velocidad más lenta que llegan a alcanzar las proteínas a la hora de ensamblar es del orden de 10 microsegundos, aunque nosotros sólo podemos simular del orden de 10 nanosegundos. Se han desarrollado nuevos caminos para simular el ensamblaje de las diferentes proteínas, los cuales pueden romper la barrera anteriormente marcada. Se basan en el procesamiento múltiple con numerosos procesadores unidos entre sí, los cuales son capaces de adquirir velocidades de proceso altísimas. De este modo, se puede romper la barrera del microsegundo y conseguir descubrir el misterio de cómo se ensamblan las diferentes proteínas entre sí.

- **Genome@Home –** Genome sequence mining: este proyecto lo está llevando a cabo el mismo grupo que he mencionado anteriormente, es decir, el Grupo Panda del Departamento de Químicas de la Universidad de Standford. Este proyecto estudia el genoma y las proteínas directamente mediante estructuras tridimensionales de la proteína. La estructura de la proteína se procesa como un conjunto de coordenadas atómicas cartesianas. Estos datos se han obtenido experimentalmente a través de rayos X o mediante técnicas NMR.

3.2.2 Pequeña historia de COSM:

El proyecto COSM nació a mediados de 1995 al mismo tiempo que empezaba la vida de los sistemas operativos, los cuales estaban diseñados en microkernel con fuertes influencias de sistemas operativos como el de Mach o como el de QNX, pero rápidamente se tornaron a los sistemas operativos distribuidos.

Una vez empezado el proyecto, el objetivo era conseguir que COSM fuera fácilmente portable a cualquier tipo de sistema operativo o CPU.

En 1997, se fundó distributed.net basándose en este proyecto al completo. En esta época se diseñaron los primeros clientes que se introdujeron en la versión 2 ("v2").

Hoy en día existe ya la versión número 3 ("v3"), versión que ha sido utilizada en el presente proyecto.

3.2.3 Objetivos del diseño:

El objetivo de COSM es construir un sistema estable, realizable y seguro para un sistema de procesamiento distribuido a gran escala.

Los Clientes sólo deberían limitarse a procesar los mandatos recibidos por el servidor y obtener una solución. Una vez obtenida ésta, devolverla. Todo esto debería realizarse sin interferir en las demás operaciones que esté realizando el sistema donde se encuentre albergado el Cliente.

Todos los comandos de consola deberían ser fáciles de manejar para el grupo de Clientes que estén corriendo en la máquina sin la necesidad, por parte del usuario, de vigilar a cada uno de estos Clientes.

Los Proxies y los Servidores deberían estar provistos de una infraestructura que les permitiera servir a una gran diversidad de Clientes.

3.2.4 Problemas existentes:

El sistema no está diseñado para atajar cualquier tipo de problema (ningún sistema puede). En muchos equipos, podemos descubrir en tan sólo 24 horas gran diversidad de problemas que implican que los equipos sean eliminados por no estar conectados, por ejemplo.

Por esta razón, COSM hace una distinción entre equipos online y equipos offline, permitiéndolos seleccionar el tipo de problema que desean solucionar.

Uno de los problemas más frecuentes que se puede presentar en un laboratorio o en un campus universitario de gran escala es el ancho de banda, el cual puede causar serios problemas como el rendering porque la red local no lo soporte. Esto se puede solucionar añadiendo más capas al proxy, de tal modo que maneje la mayor parte de los datos que

se procesan por la red, consiguiendo así la descarga de trabajo en la misma y pudiendo así ceñirnos al ancho de banda existente.

3.2.5 Seguridad:

La autenticación en este modelo está basada en el criptosistema de clave pública de Diffie-Hellman.

Todos los comandos, descripciones de proyectos, etc., deben ser certificados por el usuario o grupo que el instalador del Cliente ha autorizado para cada una de esas acciones.

El usuario puede bloquear las automodificaciones o comandos de todas aquellas personas que él crea oportunas. Esto requiere que el usuario compile sus propios núcleos y los distribuya entre sus Clientes ciñéndose siempre bajo la configuración de su proxy.

El balance entre seguridad y simplicidad siempre quedará bajo las manos del propio usuario.

3.3 Características técnicas:

Todo el código, como se ha comentado anteriormente, ha sido realizado en ANSI C. Esto nos permite que la plataforma sea fácilmente portable y haga que los usuarios no se encuentren con serios problemas a la hora de compilar el código como ellos deseen.

Tendremos una librería común para el código del Cliente, para el código del proxy y para el servidor. Nos permitirá trabajar de manera más rápida y completa a la hora de desarrollar el sistema.

La escalabilidad la suponemos de la siguiente manera:

- 100 Kb/segundo en el servidor o en un Full Proxy.
- 2 Kb de datos por paquete de trabajo.
- Cada servidor puede manejar 4.32M equipos y cada Full proxy puede manejar 864K equipos o proxies personales.
- Si de media hay 10 equipos que pueden ser manejados, 32 Full proxies son configurados y hay hasta un total de 64 servidores maestros, el sistema es capaz de manejar hasta 276M equipos.

Ésta es una previsión muy pesimista, es decir, el caso peor puesto que la estimación asumida no ha contado con la compresión y el trabajo es asignado a cada equipo a instancias de su proxy sin que se dé así el caso en la realidad. La escalabilidad debería ser fácilmente superada, tan sólo añadiendo unos cuantos Servidores y Full Proxies.

Existe dentro de la plataforma Cliente/Servidor un sistema distribuido de ficheros que nos permite almacenar los datos con una redundancia superior al 4:1.

3.4 Arquitectura de red y topología:

3.4.1 Introducción:

El principal objetivo en el diseño de la red es el asegurar que el trabajo delegado en los Clientes sea realizado.

El trabajo que se deposita dentro del sistema necesita ser completado rápida y eficientemente y los resultados tienen que ser devueltos al servidor. Una vez finalizado el trabajo, éste es eliminado del sistema y numerosos trabajos están a la espera de entrar en el sistema para ser resueltos.

Todos los nodos deberían ser capaces de comunicar la información que se necesite en cualquier momento.

El diseño es altamente redundante a todos los niveles de tal modo que, si un nodo cae, esto no debería afectar al sistema en absoluto, puesto que el resto de los nodos no se ven afectados.

Los proxies y los servidores actúan en anillo donde cualquier nodo, dentro del anillo, puede servir la misma función que cualquier otro.

La seguridad es un factor crítico dentro de cualquier sistema distribuido. Ningún nodo activará ningún comando o usará ningún dato hasta que se verifique el origen de ese comando o ese dato.

3.4.2 Arquitectura de red:

Hay tres capas en la red, que son las siguientes:

- **Servidores**
  Los servidores son el nivel más alto de la estructura y atienden en grupos de uno o más. La función de los servidores es la de manejar las peticiones, recibir los trabajos completados y la de proporcionar cualquier dato solicitado relacionado con el trabajo que se esté desempeñando. Los servidores se comunicarán con los proxies que estén contenidos en su capa o se comunicarán con otros servidores, pero nunca se comunicarán con los clientes.

- **Proxies**
  Los proxies actúan como intermediarios dentro de la red y son los que se encargan de realizar la mayoría de las tareas relacionadas con las comunicaciones. Un proxy, normalmente, hablará o se comunicará con otros proxies de su misma capa para distribuirles mensajes, cargar balances o intentar resolver problemas que se presenten dentro de la propia red.

- **Clientes**
  Los clientes son los trabajadores natos de la red. Ellos son los que reciben las peticiones de trabajo por parte de los proxies y, una vez obtenidos los resultados, se los devuelven a los proxies que consideren oportunos para que estos resultados lleguen al servidor deseado y que éste pueda procesar los datos. Los clientes han sido diseñados para que, una vez instalados, el usuario se pueda olvidar de ellos a menos que desee configurar algún aspecto de seguridad o de otro tipo. Los clientes nunca interferirán con los procesos que se encuentren lanzados en la máquina local en la que se encuentren instalados. En el caso de que al Cliente se le presente un problema que por sí solo no puede resolver, el Cliente será capaz de mandar un mensaje a su propietario indicándole esta situación para que sea resuelta.

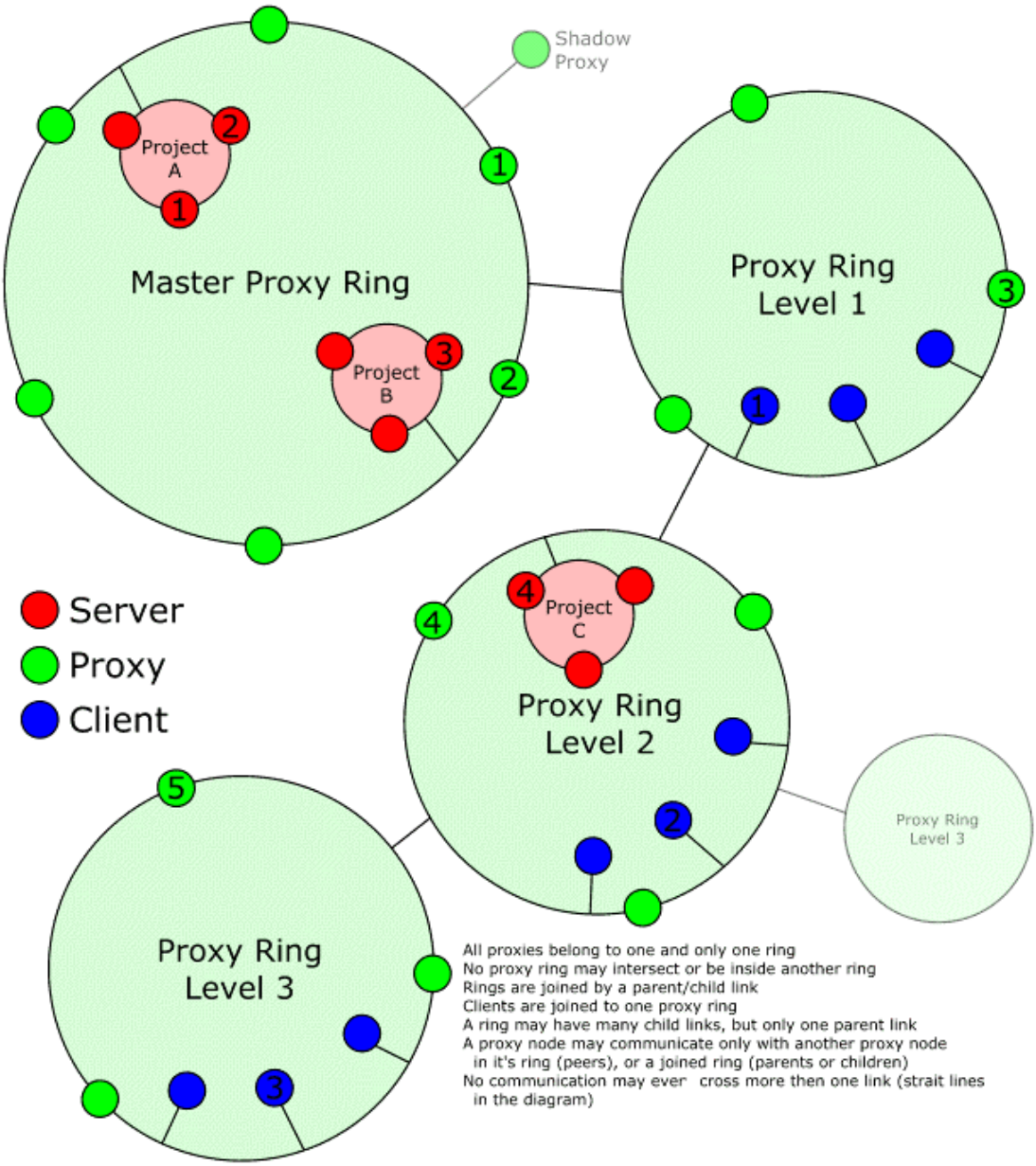
A parte de estas tres capas, también tenemos otros tres elementos que son importantes dejar constancia de ellos dentro de la topología y arquitectura de red que rodea a la plataforma Cliente/Servidor implementada:

- **Comandos de Consola**
  Los comandos de consola son el interfaz que tiene el usuario para poder comunicarse con otros elementos de la red. Con este tipo de comandos, el usuario es capaz de comandar algunos o incluso todos los nodos que el usuario tenga autorización de control. Son el modo de interactuar, por parte de los usuarios, con el sistema y sólo tienen una única forma de interactuar con el sistema. Este tipo de comandos soportan la mayoría de lenguajes naturales, es decir, el lenguaje que se encuentra más cercano al ser humano, y están estandarizados.

- **"Cores" o Núcleos**
  Los núcleos representan el trabajo real que se lleva a cabo en el sistema. Sólo manejarán aquellos trabajos que estén involucrados en el proyecto que se esté atendiendo en ese momento. De este modo, se consigue que el núcleo sea portable a cualquier procesador o sistema operativo de manera fácil y simple. Con esto también conseguimos que los núcleos sean capaces de trabajar *offline* y separar a aquellos Clientes cuyos equipos se encuentran desconectados del resto de la red que se encuentra conectada. Todos los núcleos deben ser verificados por los clientes para cerciorarse del estado del núcleo o evitar correr uno que se encuentre en mal estado.

- **Módulos servidores**
  Un módulo servidor es un elemento de cualquier servidor que se encuentre manejando peticiones de un proyecto específico. Se registran con un conjunto de librerías estándar alojadas en el servidor y estarán siempre a la disposición del servidor para ayudarle en las peticiones que se soliciten del determinado proyecto que se esté atendiendo. Son capaces de generar trabajos nuevos y, del mismo modo, son capaces de procesar y verificar los resultados obtenidos.

A continuación presentaremos un diagrama, a modo de ejemplo, para que nos podamos hacer una idea de cómo quedaría la arquitectura de red.



3.5 Convenciones generales seguidas en la implementación:

El propósito de estas convenciones generales a la hora de desarrollar el código es el de establecer una metodología común en la implementación del código desarrollado.

Hay que lograr una simplicidad y una claridad en el código para que pueda ser comprensible por terceras personas. Todo el código ha sido diseñado para que sea lo más comprensible que se pueda.

Como desde el principio se ha deseado que el código fuese portable, todo ha sido realizado bajo ANSI C.

Todo el código está libre de errores y, en cualquier sitio donde se desee compilar, deberá estar libre de errores y "warnings".

El código se encuentra comentado. Incluso simples bucles "*for*" tienen alguna línea de comentarios para que se facilite la labor de comprensión a una tercera persona. Las funciones también se encuentran correctamente comentadas al principio, en su cabecera.

Todos estos comentarios de funciones contienen lo que hace la función, parámetros requeridos, valores que se esperan obtener, o los errores que se pueden producir en la propia función.

Los nombres tanto de funciones como de variables o cualquier otro tipo de elemento utilizado tienen que hacer referencia lo que están representando y tienen que ser lo suficientemente largos para que se conviertan, a su vez, en un autocomentario.

Las variables, funciones, estructuras y cualquier otro tipo de definición empiezan con:

- v3_
- v3
- V3_
- V3

A la hora de nombrar los ficheros, tanto los .c como los .h, se ha intentado que ambos tuvieran el mismo nombre para saber quién hace referencia a quién.

Con las funciones se ha seguido la metodología de empezar la función con una letra mayúscula y la labor que realiza la función también se representa en el nombre de la función con otra letra mayúscula, por ejemplo, ReadConfig (char *nombre_fichero).

Las variables utilizadas para los bucles han sido nombradas con las típicas coordenadas cartesianas, es decir, x, y, z, etc. Para el resto de variables se pretende que sean descriptivas, por lo que los nombres se han separado con un guión bajo para las variables locales, como por ejemplo, u64 word_count.

No se ha hecho uso de **ninguna** variable global por ir en contra de la programación estructurada y porque el uso de este tipo de variables puede proporcionar resultados inesperados en cualquier momento y, por tanto, presentarse un problema difícil de resolver.

Cuando se ha pretendido que se devolviera algún tipo de resultado, se ha usado la convención de que el valor 0 significa que ha pasado, mientras que el valor distinto de 0 significa que ha fallado. De este modo, tenemos dos variables, V3_PASS y V3_FAIL, que representan estos valores. Un código ejemplo de lo que estamos comentando podría ser el siguiente:

```
if ( (error = test() ) != V3_PASS)
        { lo marcaríamos como un problema e intentaríamos recuperarlo}
```

Cuando se devuelven punteros, NULL nos indica fallo y cualquier otro valor lo consideraremos como valor válido. Un código ejemplo de esto sería:

```
if ( ( pointer = test()  ) == NULL)
        { lo marcaríamos como un problema e intentaríamos recuperarlo}
```

No se ha hecho uso de los tipos de datos int, char, short, etc. Todos los parámetros se han declarado con el formato universal, siempre pensando en la portabilidad de la plataforma a cualquier sistema operativo, puesto que se puede dar el caso de que alguno de los sistemas operativos existentes no entiendan este tipo de datos.

3.6 Documentación de funciones:

3.6.1 Introducción:

En el siguiente apartado pasaremos a exponer y explicar tanto las librerías como las funciones que forman parte de la plataforma Cliente/Servidor que se ha llevado a cabo.

Con esto se pretende tener una documentación clara y concisa de cada una de las partes del programa que conforma la plataforma expuesta.

3.6.2 Librerías:

A continuación, pasaremos a comentar las librerías más importantes de la plataforma Cliente/Servidor:

- **cosm.h**: es la librería principal de la plataforma. Incluye todas las cabeceras de las librerías de la capa de CPU/OS y de la capa de utilidades, así como las funciones de "autotest" desarrolladas.

- **Capa de CPU/OS**: ésta es la capa en la que se encuentran enmarcadas aquellas librerías concernientes a las distintas CPU´s y los distintos Sistemas operativos. Entre ellas tenemos:

  - **cputypes.h**: En ella se encuentran todas las definiciones y los tipos.
  - **cosmfile.h**: Contiene todas las declaraciones de funciones de ficheros y directorios.
  - **cosmgui.h**: Hace referencia a las funciones para el interfaz gráfico de usuario.
  - **cosmio.h**: Contiene las funciones que manejan la entrada y salida para cadenas de caracteres.
  - **cosmmath.h**: Aquí se encuentran las cabeceras de todas las funciones matemáticas necesarias para la implementación de la plataforma.
  - **cosmmem.h**: Aquí aparecen las cabeceras de todas las funciones para la gestión de memoria necesarias para la implementación de la plataforma.
  - **cosmnet.h**: En ella se encuentran las cabeceras de todas las funciones relacionadas con la gestión de la red necesarias para la implementación de la plataforma.
  - **cosmtask.h**: Aquí aparecen las cabeceras de todas las funciones relacionadas con la gestión de procesos, hilos o threads, CPU´s, semáforos o mutex y de reloj.

- **Capa de Utilidades**: ésta es la capa en la que se encuentran enmarcadas aquellas librerías concernientes a las distintas utilidades necesarias. Entre ellas tenemos:

  - **buffer.h**: En ella se encuentran las cabeceras de todas las funciones relacionadas con la gestión de la memoria de los buffers.
  - **compress.h**: Aquí destacan las cabeceras de todas las funciones relacionadas con la gestión de compresión de los datos.
  - **config.h**: Contiene las cabeceras de todas las funciones relacionadas con la configuración de los distintos aspectos del programa.
  - **cosmtime.h**: Hace referencia a las cabeceras de todas las funciones relacionadas con la gestión del tiempo.
  - **dfs.h**: Aquí destacan las cabeceras de todas las funciones relacionadas con la gestión de los ficheros distribuidos.
  - **email.h**: Hacen referencia a las cabeceras de todas las funciones relacionadas con la gestión de los E-Mail.
  - **http.h**: En ella se encuentran las cabeceras de todas las funciones relacionadas con la gestión del protocolo http (Hypertext Transfer Protocol o Protocolo para la Transferencia de Hipertexto).
  - **language.h**: Aquí destacan las cabeceras de todas las funciones relacionadas con la gestión del lenguaje natural, es decir, el lenguaje máquina más próximo al ser humano.
  - **log.h**: Hace referencia a las cabeceras de todas las funciones relacionadas con la gestión de los ficheros de log.
  - **security.h**: Contiene las cabeceras de todas las funciones relacionadas con la gestión de la firma de los datos, cifrado, funciones hash y funciones aleatorias.

3.6.3 Índice por orden alfabético de Funciones:

En este apartado mostraremos un índice de todas las funciones existentes dentro de la plataforma Cliente/Servidor para que el lector pueda localizar cualquier tipo de función que desee consultar más rápidamente.

Las funciones se encuentran ordenadas alfabéticamente. El índice es el siguiente:

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

Continuación S

T

U

Continuación U

V

W

X

Y

Z

3.6.4 Funciones implementadas:

En este apartado pasaremos a mostrar la sintaxis de todas las funciones implementadas, así como una pequeña descripción de las mismas, los valores que pueden devolver, los errores que se pueden producir en ellas y un pequeño ejemplo descriptivo. También mostraremos las definiciones y los tipos empleados.

Las funciones las vamos a reunir en 15 grandes grupos para que queden emparejadas dependiendo de la funcionalidad de las mismas, más un gran grupo en el que se presentarán todas las definiciones y tipo utilizados. Los 16 grandes grupos de los que estamos hablando son los siguientes:

- Funciones para Procesos, hilos o threads, CPU y semáforos o mutex.
- Funciones matemáticas.
- Funciones para la memoria del Buffer.
- Funciones para la compresión de datos.
- Funciones para la configuración de los ficheros del programa.
- Funciones de ficheros y directorios.
- Función para E-Mail.
- Funciones para la entrada y salida de cadenas de caracteres.
- Funciones para la firma de datos, cifrado y funciones hash.
- Funciones para los ficheros log.
- Funciones para la memoria.
- Funciones de red.
- Funciones para sistema “Benchmark”.
- Funciones para el tiempo.
- Funciones del programa principal y de “autotest”.
- Definiciones y tipos.

3.6.4.1 Funciones para Procesos, hilos o threads, CPU y semáforos o mutex:

❋ **v3ProcessID:**

- Sintaxis:

```
#include "cosmtask.h"
u64 v3ProcessID( void );
```

- Descripción:

  Obtener el ID del proceso actual.

- Valores devueltos:

  El ID del proceso actual ó 0 para indicar que se ha producido un error.

- Errores:

  La posible causa de fallo es la siguiente:
  - El sistema operativo no tiene procesos.

- Ejemplo:

```
u64 pid;
pid = v3ProcessID();
```

❋ **v3ProcessPriority:**

- Sintaxis:

```
#include "cosmtask.h"
u8 v3ProcessPriority( u8 priority );
```

- Descripción:

  Establecer la prioridad del proceso actual. Si la variable *priority* vale 0, no se realizan cambios. De otro modo, la prioridad se almacenará en la variable anteriormente comentada. Las prioridades se encuentran dentro del rango que va desde el 1 hasta el 255. No se puede incrementar la prioridad cuando nos encontremos en el valor por defecto 255. No todos los sistemas operativos nos permitirán devolver una prioridad más alta si ésta ha sido reducida con anterioridad.

- Valores devueltos:

  La prioridad del proceso, la cual se encontrará entre 1 y 255 después que la llamada al procedimiento se haya completado, o se devolverá 0 para indicar que se ha producido un error.

- Errores:

  Las posibles causas de fallo son las siguientes:
  - El sistema operativo no soporta esta función.
  - El sistema operativo no permite incrementar la prioridad de un proceso una vez que a éste se le ha fijado una prioridad más baja.

- Ejemplo:

```
u8 priority;
u8 new_priority;
priority = v3ProcessPriority( (u8) 0 );
if ( priority < 100 )
        priority =+ 40;
new_priority = v3ProcessPriority( priority );
if ( new_priority == 0 )
        /* Error */
```

❋ **v3ProcessSpawn:**

- Sintaxis:

```
#include "cosmtask.h"
s32 v3ProcessSpawn( u64 * process_id, const ascii * command,
                    const ascii * arguments, ... );
```

- Descripción:

  Monitoriza a un nuevo proceso que está corriendo y se almacena en la variable *command*. Esta variable debe tener almacenado un path en formato binario. La variable *process_id* es la variable que almacena el ID del proceso que estamos tratando. Y, por último, la variable *arguments* debe finalizar con un NULL.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Las posibles causas de fallo son las siguientes:
  - El sistema operativo no soporta esta función.
  - Que el path que hayamos pasado sea incorrecto.
  - Que el sistema carezca de procesos, no haya memoria, etc.

- Ejemplo:

```
u64 child;
if ( v3ProcessSpawn( &child, "cat", "file.txt","file2.txt", NULL ) !=
V3_PASS )
{
  v3PrintA( (ascii *) "Spawn Failed\n" );
}
else
{
  v3PrintA( (ascii *) "Child PID = %v\n", child );
}
```

❋ **v3ProcessSignal:**

- Sintaxis:

```
#include "cosmtask.h"
s32 v3ProcessSignal( u64 process_id, u32 signal );
```

- Descripción:

Envía el contenido de la variable *signal* al proceso cuyo ID se almacena en la variable *process_id*.

- Señales:

V3_PROCESS_SIGPING: test para la existencia de procesos.

V3_PROCESS_SIGINT: guardar o checkpoint.

V3_PROCESS_SIGTERM: guardar o terminar.

V3_PROCESS_SIGKILL: terminar con extremo prejuicio.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

La posible causa de fallo es la siguiente:

- Señal desconocida.

- Ejemplo:

```
u64 pid;
s32 result;
pid = v3ProcessID();
result = v3ProcessSignal( pid, (u32) V3_SIGNAL_PING );
if ( result != V3_PASS )
  {
   /* Target process didn't exist */
  }
```

❋ **v3ProcessEnd:**

- Sintaxis:

```
#include "cosmtask.h"
void v3ProcessEnd( int status );
```

- Descripción:

  Finaliza el proceso actual junto con todos los hilos o threads que tiene asociados. Utiliza la variable *status* para almacenar el código de salida.

- Valores devueltos:

  Ninguno.

- Errores:

  Ninguno.

- Ejemplo:

  V3ProcessEnd(1);

❋ **v3CPUCount:**

- Sintaxis:

```
#include "cosmtask.h"
s32 v3CPUCount( u32 * count );
```

- Descripción:

  Obtiene el número de CPU´s que se encuentran en el sistema y lo almacena en la variable *count*. También se puede producir un fallo, de manera que la función nos indica que el sistema no ha detectado ninguna CPU.

- Valores devueltos:

  El número de CPU´s que hay en el sistema. Siempre tiene que devolver 1 o más. La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Las posibles causas de fallo son las siguientes:
  - El sistema operativo no soporta SMP.
  - El sistema operativo no está provisto de un sistema que cuente el número de CPU´s que se encuentran asociadas al sistema.

- Ejemplo:

```
u32 num_cpus;
if ( v3CPUCount( &num_cpus ) == V3_FAIL )
{
   /* error, but num_cpus is 1 */
 }
```

❋ **v3CPUGet:**

- Sintaxis:

```
#include "cosmtask.h"
s32 v3CPUGet( u32 * cpu );
```

- Descripción:

Obtiene el número de CPU en la que se encuentra lanzado un proceso y lo almacena en la variable *cpu*. Esto no tiene un significado real si la CPU se encuentra bloqueada pero, de este modo, podemos evitar que un proceso se mueva si hemos llamado con anterioridad a la función V3CPULock.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

La posible causa de fallo es la siguiente:

- El sistema operativo no soporta SMP.

- Ejemplo:

```
u32 cpunum;
s32 result;
result = v3CPUGet( &cpunum );
if ( result != V3_PASS )
{
   /* Error */
}
```

❋ **v3CPULock:**

- Sintaxis:

```
#include "cosmtask.h"
s32 v3CPULock( u64 process_id, u32 cpu );
```

- Descripción:

Bloquea al proceso cuyo ID se encuentra almacenado en la variable *process_id* y a cualquier hilo o thread para la CPU cuyo número se encuentra almacenado en la variable *cpu*. Esta función es primordial para la implementación de la gestión de memoria V3_MEM_SECURE.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

La posible causa de fallo es la siguiente:

- El sistema operativo no soporta SMP.

- Ejemplo:

```
u64 pid;
u32 cpunum;
s32 result;
pid = v3ProcessID();
if ( pid == 0 )
{
   /* Error */
}
result = v3CPUGet( &cpunum );
if ( result != V3_PASS )
{
   /* Error */
}
result = v3CPULock( pid, cpunum );
if ( result != V3_PASS )
{
   /* Error */
 }
```

❋ **v3CPUUnlock:**

- Sintaxis:

```
#include "cosmtask.h"
s32 v3CPUUnlock( u64 process_id );
```

- Descripción:

Desbloquea al proceso cuyo ID se encuentra almacenado en la variable *process_id* de una CPU. Si se está usando el sistema de memoria V3_MEM_SECURE, la llamada a esta función no es aconsejable.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

La posible causa de fallo es la siguiente:

- El sistema operativo no soporta SMP.

- Ejemplo:

```
u64 pid;
u32 cpunum;
s32 result;
pid = v3ProcessID();
if ( pid == 0 )
  {
    /* Error */
  }
result = v3CPUGet( &cpunum );
if ( result != V3_PASS )
{
    /* Error */
}
result = v3CPULock( pid, cpunum );
if ( result != V3_PASS )
{
    /* Error */
}
result = v3CPUUnlock( pid );
if ( result != V3_PASS )
{
    /* Error */
 }
```

❋ **v3ThreadBegin:**

- Sintaxis:

```
#include "cosmtask.h"
s32 v3ThreadBegin( u64 * thread_id, void (*start)(void *),
                   void * arg, u32 stack_size );
```

- Descripción:

Comienza a utilizar un thread con argumentos almacenados en la variable *arg*, con el tamaño de la pila establecido en la variable *stack_size* y el número de ID del thread almacenado en la variable *thread_id*.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

Las posibles causas de fallo son las siguientes:

- Que el sistema se haya compilado sin el soporte a threads.
- Que el sistema tenga limitados los threads.
- Memoria limitada.

- Ejemplo:

```
void thread_function( void * arg )
{
        u32 * puppy;
        puppy = (u32 *) arg;
        /* thread code */
}
/* ... */
u64 thread_id;
u32 data;
data = 42;
if ( v3ThreadBegin( &thread_id, thread_function,
  (void *) &data, 4096 ) != V3_PASS )
{
   /* thread failed to start */
 }
```

❋ **v3ThreadID:**

- Sintaxis:

```
#include "cosmtask.h"
u64 v3ThreadID( void );
```

- Descripción:

Obtiene el ID del thread actual.

- Valores devueltos:

La función nos devolverá el ID del thread actual ó 0 en caso de que se produjera un error.

- Errores:

La posible causa de fallo es la siguiente:

- Que el sistema se haya compilado sin el soporte a threads.

- Ejemplo:

```
u64 thread_id;
thread_id = v3ThreadID();
```

❋ **v3ThreadPriority:**

- Sintaxis:

```
#include "cosmtask.h"
u8 v3ThreadPriority( u8 priority );
```

- Descripción:

Establecer la prioridad del thread actual. Si la variable *priority* vale 0, no se realizan cambios. De otro modo, la prioridad se almacenará en la variable anteriormente comentada. La prioridades se encuentran dentro del rango que va desde el 1 hasta el 255. No se puede incrementar la prioridad cuando nos encontremos en el valor por defecto (255). No todos los sistemas operativos nos permitirán devolver una prioridad más alta si ésta ha sido reducida con anterioridad.

- Valores devueltos:

La prioridad del thread, la cual se encontrará entre 1 y 255 después que la llamada al procedimiento se haya completado o se devolverá 0 para indicar que se ha producido un error.

- Errores:

  Las posibles causas de fallo son las siguientes:
  - El sistema operativo no soporta esta función.
  - Que el sistema se haya compilado sin el soporte a threads.

- Ejemplo:

```
u8 priority;
u8 new_priority;
priority = v3ThreadPriority( (u8) 0 );
if ( priority < 100 )
{
   priority =+ 40;
}
new_priority = v3ThreadPriority( priority );
if ( new_priority == 0 )
{
   /* Error */
 }
```

❋ **v3ThreadEnd:**

- Sintaxis:

```
#include "cosmtask.h"
 void v3ThreadEnd(void);
```

- Descripción:

  Finaliza la llamada del thread.

- Valores devueltos:

  Ninguno.

- Errores:

  Ninguno.

- Ejemplo:

  V3ProcessEnd( );

❋ **v3MutexLock:**

- Sintaxis:

```
#include "cosmtask.h"
s32 v3MutexLock( v3_MUTEX * mutex, u32 wait );
```

- Descripción:

Establece un bloqueo exclusivo. Nunca debe darse el caso de que exista más de un bloqueo por thread a la vez, ya que esto puede provocar resultados inesperados. Es conveniente indicar que, cada vez que se cree un Mutex, éste debe ser liberado a través de la función v3MutexFree.

- Modos de espera:

V3_MUTEX_WAIT: no devuelve nada mientras que el bloqueo esté disponible.

V3_MUTEX_NOWAIT: devuelve valores incluso si el bloqueo no estuviera disponible.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

La posible causa de fallo es la siguiente:

- Que el valor de la variable *wait* no sea adecuado.

- Ejemplo:

```
v3_MUTEX lock;
if ( v3MutexLock( &lock, V3_MUTEX_WAIT ) != V3_PASS )
{
   /* bad error */
}
/* do critical work */
v3MutexUnlock( &lock );
while ( v3MutexLock( &lock, V3_MUTEX_NOWAIT ) != V3_PASS )
{
   /* mutex was already locked, do something */
}
/* do critical work */
v3MutexUnlock( &lock );
```

❋ **v3MutexUnlock:**

- Sintaxis:

```
#include "cosmtask.h"
s32 v3MutexUnlock( v3_MUTEX * mutex );
```

- Descripción:

  Elimina un bloqueo exclusivo.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Ninguno.

- Ejemplo:

```
v3_MUTEX lock;
if ( v3MutexLock( &lock, V3_MUTEX_WAIT ) != V3_PASS )
{
   /* bad error */
}
/* do critical work */
v3MutexUnlock( &lock );
```

❋ **v3MutexFree:**

- Sintaxis:

```
#include "cosmtask.h"
void v3MutexFree( v3_MUTEX * mutex );
```

- Descripción:

  Libera al mutex cuyo ID de mutex se almacena en la variable *mutex* y los recursos del sistema relacionados.

- Valores devueltos:

  Ninguno.

- Errores:

  Ninguno.

❋ **v3Sleep:**

- Sintaxis:

```
#include "cosmtask.h"
u32 v3Sleep( u32 millisec );
```

- Descripción:

"Duerme" al sistema durante la cantidad de milisegundos almacenados en la variable *millisec*.

- Valores devueltos:

La función nos devolverá 0 si el tiempo de descanso ha sido completado, o nos devolverá el tiempo que nos queda de descanso.

- Errores:

Ninguno.

- Ejemplo:

```
u32 sleeptime;
u32 result;
sleeptime = 2300;
result = v3Sleep( sleeptime );
if ( result != 0 )
{
   /* Didn't finish sleeping */
 }
```

❋ **v3Yield:**

- Sintaxis:

```
#include "cosmtask.h"
void v3Yield( void );
```

- Descripción:

Cede a otro proceso o thread aquello que se quiera lanzar.

- Valores devueltos:

Ninguno.

- Errores:

Ninguno.

- Ejemplo:

  v3Yield( );

❋ **v3SignalRegister:**

- Sintaxis:

  #include "cosmtask.h"
  s32 v3SignalRegister(u32 signal_type, void (*handler)(int));

- Descripción:

  El registro, almacenado en la variable *handler*, actúa como manejador para la variable *signal_type*. La variable anteriormente mencionada debe tener uno de estos dos valores: o bien V3_PROCESS_SIGINT, o bien V3_PROCESS_SIGTERM. Se recomienda ver la función v3ProcessSignal para comprender mejor el envío de señales. Una vez que el manejador es lanzado por el sistema operativo, se debe reconfigurar al manejador de tal modo que, si se decide reutilizar alguna señal, esto debe ser registrado en el manejador. Hay que intentar que la duración del manejador sea lo más corta posible.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Las posibles causas de fallo son las siguientes:
  - Desconocimiento de la señal.
  - El sistema operativo no soporta esta función

- Ejemplo:

```
void signal_function( int arg )
{
/* signal code */
v3SignalRegister( V3_SIGNAL_INT, signal_function );
/* set a flag or other handler actions */
}
/* ... */
if ( v3SignalRegister( V3_SIGNAL_INT, signal_function )!= V3_PASS )
{
/* error */
}
```

❋ **v3SystemClock:**

- Sintaxis:

```
#include "cosmtask.h"
s32 v3SystemClock( v3_time * local_time );
```

- Descripción:

Obtiene la hora del sistema y la almacena en la variable *local_time*. *v3_time* es un número con signo (s128) de segundos en un formato de punto fijo basado en el tiempo 0 que es igual a: 00:00:00 UTC, Jan 1, 2000 AD.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

La posible causa de fallo es la siguiente:

- El sistema no dispone del hardware necesario para el reloj.

- Ejemplo:

```
v3_time clock;
if ( v3SystemClock( &clock ) != V3_PASS )
{
   /* temporal rift */
 }
```

3.6.4.2 Funciones matemáticas:

❋ **v3{bigger}{smaller}:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64u32( u32 a );
s64 v3s64s32( s32 a );
u128 v3u128u32( u32 a );
s128 v3s128s32( s32 a );
u128 v3u128u64( u64 a );
s128 v3s128s64( s64 a );
```

- Descripción:

Convierte un entero en un entero largo, pero no puede convertir un número con signo en un número sin signo ni viceversa.

- Valores devueltos:

  La función nos devolverá el entero más largo.

- Errores:

  Ninguno.

- Ejemplo:

```
u32 cat = 39;
u64 tiger;
tiger = v3u64u32( cat );
```

❋ **v3{smaller}{bigger}:**

- Sintaxis:

```
#include "cosmmath.h"
u32 v3u32u64( u64 a );
s32 v3s32s64( s64 a );
u32 v3u32u128( u128 a );
s32 v3s32s128( s128 a );
u64 v3u64u128( u128 a );
s64 v3s64s128( s128 a );
```

- Descripción:

  Convierte un entero en un entero de menor tamaño. Por esta razón, se puede dar el caso de que se corte el número (Truncate). No se puede convertir un número con signo en un número sin signo ni viceversa.

- Valores devueltos:

  La función nos devolverá el entero más pequeño.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 pig;
u32 ham;
_V3_SET64( pig, FEDCBA98, 76543210 );
ham = v3u32u64( pig );
/* ham = 0x76543210 */
```

❋ **v3{unsigned}{signed}:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64s64( s64 a ); /* u64 <- s64 */
u128 v3u128s128( s128 a ); /* u128 <- s128 */
```

- Descripción:

  Convierte un entero con signo en un entero sin signo del mismo tamaño. El signo puede cambiar.

- Valores devueltos:

  La función nos devolverá el entero sin signo.

- Errores:

  Ninguno.

- Ejemplo:

```
s64 a;
u64 b;
a = v3s64s32( -39 );
b = v3u64s64( a );
/* b = 0xFFFFFFFFFFFFFFD9 */
```

❋ **v3{signed}{unsigned}:**

- Sintaxis:

```
#include "cosmmath.h"
s64 v3s64u64( u64 a ); /* s64 <- u64 */
s128 v3s128u128( u128 a ); /* s128 <- u128 */
```

- Descripción:

  Convierte un entero sin signo en un entero con signo del mismo tamaño. El signo puede cambiar.

- Valores devueltos:

  La función nos devolverá el entero con signo.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 a;
s64 b;
_V3_SET64( a, FFFFFFFF, FFFFFFFF );
b = v3s64u64( a );
/* b = -1 */
```

❋ **v3Add:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Add( u64 a, u64 b );
s64 v3s64Add( s64 a, s64 b );
u128 v3u128Add( u128 a, u128 b );
s128 v3s128Add( s128 a, s128 b );
```

- Descripción:

  Suma dos enteros y nos devuelve el resultado.

- Valores devueltos:

  La función nos devolverá la suma de los dos argumentos.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 expenses = v3u64u32( 45 );
u64 fees = v3u64u32( 102 );
u64 total_costs;
total_costs = v3u64Add( fees, expenses );
```

❋ **v3Sub:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Sub( u64 a, u64 b );
s64 v3s64Sub( s64 a, s64 b );
u128 v3u128Sub( u128 a, u128 b );
s128 v3s128Sub( s128 a, s128 b );
```

- Descripción:

  Resta dos enteros y nos devuelve el resultado.

- Valores devueltos:

  La función nos devolverá la resta de los dos argumentos.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 income = v3u64u32( 235 );
u64 expenses = v3u64u32( 19 );
u64 net;
net = v3u64Sub( income, expenses );
```

❋ **v3Mul:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Mul( u64 a, u64 b );
s64 v3s64Mul( s64 a, s64 b );
u128 v3u128Mul( u128 a, u128 b );
s128 v3s128Mul( s128 a, s128 b );
```

- Descripción:

  Multiplica dos enteros y nos devuelve el resultado.

- Valores devueltos:

  La función nos devolverá la multiplicación de los dos argumentos.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 length = v3u64u32( 7 );
u64 width = v3u64u32( 13 );
u64 area;
area = v3u64Mul( length, width );
```

❋ **v3Div:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Div( u64 a, u64 b );
s64 v3s64Div( s64 a, s64 b );
u128 v3u128Div( u128 a, u128 b );
s128 v3s128Div( s128 a, s128 b );
```

- Descripción:

  Divide dos enteros y nos devuelve el resultado.

- Valores devueltos:

  La función nos devolverá la división de los dos argumentos.

- Errores:

  Si la variable *b* vale 0, el programa está definido para que se evite una división por 0 y, por tanto, la función nos devolverá 0.

- Ejemplo:

```
u64 candies = v3u64u32( 100 );
u64 children = v3u64u32( 5 );
u64 candies_each;
candies_each = v3u64Div( candies, children );
```

❋ **v3Mod:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Mod( u64 a, u64 b );
s64 v3s64Mod( s64 a, s64 b );
u128 v3u128Mod( u128 a, u128 b );
s128 v3s128Mod( s128 a, s128 b );
```

- Descripción:

  Nos da el resultado de la operación *a* mod *b*.

- Valores devueltos:

  La función nos devolverá o bien 0, o bien |b|-1.

- Errores:

  Si la variable *b* vale 0, el programa esté definido para que se evite una división por 0 y, por tanto, la función nos devolverá 0.

- Ejemplo:

```
u64 cards = v3u64u32( 52 );
u64 players = v3u64u32( 7 );
u64 leftover_cards;
leftover_cards = v3u64Mod( cards, players );
```

❋ **v3Inc:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Inc( u64 * a );
s64 v3s64Inc( s64 * a );
u128 v3u128Inc( u128 * a );
s128 v3s128Inc( s128 * a );
```

- Descripción:

  Incrementa en una unidad el entero almacenado en la variable *a* por referencia.

- Valores devueltos:

  La función nos devolverá la variable *a* incrementada en una unidad. Este valor es equivalente a hacer *a*++.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 i, j;
_V3_SET64( i, 01234567, 89ABCDEF );
j = v3u64Inc( &i );
/*
  i is now 0x0123456789ABCDF0
  and j is 0x0123456789ABCDEF
  same as: j = i++;
*/
```

❋ **v3Dec:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Dec( u64 * a );
s64 v3s64Dec( s64 * a );
u128 v3u128Dec( u128 * a );
s128 v3s128Dec( s128 * a );
```

- Descripción:

Decrementa en una unidad el entero almacenado en la variable *a* por referencia.

- Valores devueltos:

La función nos devolverá la variable *a* decrementada en una unidad. Este valor es equivalente a hacer *a*--.

- Errores:

Ninguno.

- Ejemplo:

```
u64 i;
_V3_SET64( i, 13845A34, 09AB4DEF );
j = v3u64Dec( &i );
/*
  i is now 0x0123456789ABCDEE
  and j is 0x0123456789ABCDEF
  same as: j = i--;
*/
```

❋ **v3Eq:**

- Sintaxis:

```
#include "cosmmath.h"
u32 v3u64Eq( u64 a, u64 b );
u32 v3s64Eq( s64 a, s64 b );
u32 v3u128Eq( u128 a, u128 b );
u32 v3s128Eq( s128 a, s128 b );
```

- Descripción:

Comprueba la igualdad de dos enteros pasados como parámetros.

- Valores devueltos:

La función nos devolverá 1 si los números son iguales y 0 en caso contrario.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 bytes_received = v3u64u32( 45 );
u64 bytes_sent = v3u64u32( 39 );
u32 all_sent;
all_sent = v3u64Eq( bytes_received, bytes_sent );
```

❋ **v3Gt:**

- Sintaxis:

```
#include "cosmmath.h"
u32 v3u64Gt( u64 a, u64 b );
u32 v3s64Gt( s64 a, s64 b );
u32 v3u128Gt( u128 a, u128 b );
u32 v3s128Gt( s128 a, s128 b );
```

- Descripción:

  Comprueba si el primer entero que se ha pasado como parámetro es mayor que el segundo.

- Valores devueltos:

  La función nos devolverá 1 si el primer argumento es más grande que el segundo y 0 en caso contrario.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 chairs = v3u64u32( 45 );
u64 people = v3u64u32( 39 );
u32 enough_chairs;
enough_chairs = v3u64Gt( chairs, people );
/* enough_chairs = 1 */
```

❋ **v3Lt:**

- Sintaxis:

```
#include "cosmmath.h"
u32 v3u64Lt( u64 a, u64 b );
u32 v3s64Lt( s64 a, s64 b );
u32 v3u128Lt( u128 a, u128 b );
u32 v3s128Lt( s128 a, s128 b );
```

- Descripción:

  Comprueba si el primer entero que se ha pasado como parámetro es más pequeño que el segundo.

- Valores devueltos:

  La función nos devolverá 1 si el primer argumento es más pequeño que el segundo y 0 en caso contrario.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 uv_rays = v3u64u32( 27 );
u64 max = v3u64u32( 46 );
u32 no_sunburn;
no_sunburn = v3u64Lt( uv_rays, max );
/* no_sunburn = 1 */
```

❋ **v3And:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64And( u64 a, u64 b );
u128 v3u128And( u128 a, u128 b );
```

- Descripción:

  Realiza la operación lógica AND entre los dos argumentos que se le ha pasado como parámetro.

- Valores devueltos:

  La función nos devolverá la operación AND realizada entre los dos parámetros.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 foo = v3u64u32( 0x0000001C );
u64 bar = v3u64u32( 0x000001E3 );
u64 go;
go = v3u64And( foo, bar );
```

❋ **v3Or:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Or( u64 a, u64 b );
u128 v3u128Or( u128 a, u128 b );
```

- Descripción:

  Realiza la operación lógica OR entre los dos argumentos que se le ha pasado como parámetro.

- Valores devueltos:

  La función nos devolverá la operación OR realizada entre los dos parámetros.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 foo = v3u64u32( 0x000002E3 );
u64 bar = v3u64u32( 0x0000008F );
u64 go;
go = v3u64Or( foo, bar );
```

❋ **v3Xor:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Xor( u64 a, u64 b );
u128 v3u128Xor( u128 a, u128 b );
```

- Descripción:

  Realiza la operación lógica XOR entre los dos argumentos que se le ha pasado como parámetro.

- Valores devueltos:

  La función nos devolverá la operación XOR realizada entre los dos parámetros.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 foo = v3u64u32( 0x021C4F19 );
u64 bar = v3u64u32( 0x00074B31 );
u64 go;
go = v3u64Xor( foo, bar );
```

❋ **v3Not:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Not( u64 a );
u128 v3u128Not( u128 a );
```

- Descripción:

  Realiza la operación lógica NOT en el argumento que se le ha pasado como parámetro.

- Valores devueltos:

  La función nos devolverá la operación NOT realizada en el parámetro.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 foo = v3u64u32( 0x00F0401C );
u64 notfoo;
notfoo = v3u64Not( foo );
```

❋ **v3Lsh:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Lsh( u64 a, u32 x );
u128 v3u128Lsh( u128 a, u32 x );
```

- Descripción:

  Mueve a la izquierda, en el entero especificado como parámetro, el número de dígitos indicados en la variable *x*.

- Valores devueltos:

  La función nos devolverá el entero permutado.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 foo = v3u64u32( 0x300E481C );
u64 fooshifted;
fooshifted = v3u64Lsh( foo, (u32) 4 );
/* fooshifted = 0x0000000300E481C0 */
```

❋ **v3Rsh:**

- Sintaxis:

```
#include "cosmmath.h"
u64 v3u64Rsh( u64 a, u32 x );
u128 v3u128Rsh( u128 a, u32 x );
```

- Descripción:

  Mueve a la derecha, en el entero especificado como parámetro, el número de dígitos indicados en la variable *x*.

- Valores devueltos:

  La función nos devolverá el entero permutado.

- Errores:

  Ninguno.

- Ejemplo:

```
u64 foo = v3u64u32( 0x000F061C );
u64 fooshifted;
fooshifted = v3u64Rsh( foo, (u32) 4 );
/* fooshifted = 0x000000000000F061 */
```

❋ **v3f64NaN:**

- Sintaxis:

```
#include "cosmmath.h"
s32 v3f64NaN( f64 number );
```

- Descripción:

  Comprueba si un número es NAN.

- Valores devueltos:

  La función nos devolverá 1 si el número que estamos comprobando es NAN y 0 en caso contrario.

- Errores:

  Ninguno.

❋ **v3f64Inf:**

- Sintaxis:

```
#include "cosmmath.h"
s32 v3f64Inf( f64 number );
```

- Descripción:

  Comprueba si un número es más o menos Infinito.

- Valores devueltos:

  La función nos devolverá 1 si el número que estamos comprobando es + Infinito, -1 si el número es - Infinito y 0 en el resto de los casos.

- Errores:

  Ninguno.

3.6.4.3 Funciones para la memoria del Buffer:

❋ **v3BufferCreate:**

- Sintaxis:

```
#include "buffer.h"
s32 v3BufferCreate( v3_BUFFER * buffer, u64 size, u32 mode,
                    u64 grow, const void * data, u64 length );
```

- Descripción:

  Crea un nuevo buffer almacenándolo en la variable *buffer*. Inicialmente, el tamaño que se asigna al buffer es el que marca la variable *size*. Una vez que el buffer se ha llenado, se incrementa la memoria con grupos de bytes cuyo tamaño viene determinado por la variable *grow*. Si esta variable vale 0, el buffer nunca llegará a crecer, por lo que se puede llegar a llenar. Si la variable *data* tiene un distinto de NULL, entonces el tamaño de bytes, determinado en la variable *length*, se copiará de la variable *data* y dentro de un nuevo buffer. La variable *mode* nos indica el tipo del buffer que se podría llegar a crear.

- Modos:

  V3_BUFFER_MODE_QUEUE: Crea un buffer con estructura de cola. Como ya sabemos, una cola tiene una estructura FIFO, es decir, el primero que entra es el primero que sale; esto quiere decir que el primer elemento que se añade al buffer será el primero en ser retirado del mismo.

  V3_BUFFER_MODE_STACK: Crea un buffer con estructura de pila. Como ya sabemos, una pila tiene una estructura LIFO, es decir, el último que entra es el primero que sale; esto quiere decir que el último elemento que se añade al buffer será el primero en ser retirado del mismo.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_BUFFER_ERROR_MEMORY: No hay suficiente memoria.

  V3_BUFFER_ERROR_MODE: Modo desconocido.

  V3_BUFFER_ERROR_FULL: Buffer ya creado.

- Ejemplo:

```
v3_BUFFER * buf;
u64 size, grow;
if ( ( buf = v3MemAlloc( v3u64u32( sizeof( v3_BUFFER ) ),
     V3_MEM_NORMAL ) ) == NULL )
{
   /* Error in memory allocation. */
   return( V3_FAIL );
}
_V3_SET64( size, 00000000, 00000400 );
_V3_SET64( grow, 00000000, 00000200 );
if ( v3BufferCreate( buf, size, V3_BUFFER_MODE_QUEUE, grow, NULL,
     v3u64u32( 0 ) ) == V3_FAIL )
{
   /* Failed */
   return( V3_FAIL );
}
```

❋ **v3BufferLength:**

- Sintaxis:

```
#include "buffer.h"
u64 v3BufferLength( v3_BUFFER * buffer );
```

- Descripción:

Proporciona el tamaño de los datos almacenados en el buffer cuyo identificador se encuentra almacenado en la variable *buffer*.

- Valores devueltos:

El tamaño de los datos almacenados en el buffer.

- Errores:

Ninguno.

- Ejemplo:

```
v3_BUFFER * buf;
u64 size, grow;
u64 length;
if ( ( buf = v3MemAlloc( v3u64u32( sizeof( v3_BUFFER ) ),
     V3_MEM_NORMAL ) ) == NULL )
{
   /* Error in memory allocation. */
   return( V3_FAIL );
}
_V3_SET64( size, 00000000, 00000400 );
_V3_SET64( grow, 00000000, 00000200 );
```

```
if ( v3BufferCreate( buf, size, V3_BUFFER_MODE_QUEUE, grow, NULL,
     v3u64u32( 0 ) ) == V3_FAIL )
{
   /* Failed */
   return( V3_FAIL );
}
length = v3BufferLength( buf ); /* Should be zero. */
```

❋ **v3BufferClear:**

- Sintaxis:

```
#include "buffer.h"
s32 v3BufferClear( v3_BUFFER * buffer );
```

- Descripción:

  Establece el tamaño del buffer (cuyo indicador viene almacenado en la variable *buffer*) a 0 sin liberarlo.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se puede dar la siguiente salida como código de error:

  V3_BUFFER_ERROR_MEMORY: identificador de buffer erróneo.

- Ejemplo:

```
v3_BUFFER * buf;
u64 length;
/* ... */
v3BufferClear( buf );
length = v3BufferLength( buf ); /* Should be zero. */
```

❋ **v3BufferPut:**

- Sintaxis:

```
#include "buffer.h"
s32 v3BufferPut( v3_BUFFER * buffer, const void * data, u64 length );
```

- Descripción:

Inserta el tamaño (*legnth*) de los datos (*data*) dentro del buffer (*buffer*).

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_BUFFER_ERROR_MEMORY: No hay suficiente memoria para realizar la operación.

V3_BUFFER_ERROR_FULL: El buffer se encuentra lleno. Esta posibilidad se da si el parámetro *grow*, a la hora de crear el buffer con la función v3BufferCreate, vale 0.

- Ejemplo:

```
v3_BUFFER * buf;
u64 size, grow;
u32 data, counter, data;
if ( ( buf = v3MemAlloc( v3u64u32( sizeof( v3_BUFFER ) ),
       V3_MEM_NORMAL ) ) == NULL )
{
   /* Error in memory allocation. */
   return( V3_FAIL );
}
_V3_SET64( size, 00000000, 00000400 );
_V3_SET64( grow, 00000000, 00000200 );
if ( v3BufferCreate( buf, size, V3_BUFFER_MODE_QUEUE, grow, NULL,
     v3u64u32( 0 ) ) == V3_FAIL )
{
   /* Failed */
   return( V3_FAIL );
}
/* Lets insert some u32's into the buffer. They are 4 bytes each. */
for ( counter = 0; counter < 20; counter++ )
{
   data = counter * counter;
   if ( v3BufferPut( buf, &data, sizeof( u32 ) ) != V3_PASS )
```

```
    {
        /* Error. */
        return( V3_FAIL );
    }
}
```

❋ **v3BufferGet:**

- Sintaxis:

```
#include "buffer.h"
u64 v3BufferGet( void * data, u64 length, v3_BUFFER * buffer );
```

- Descripción:

Obtiene tantos bytes como sea posible, poniendo al máximo de longitud (*length*) el buffer (*buffer*) colocándolo dentro de la variable *data*.

- Valores devueltos:

La función nos devolverá el tamaño, es decir, el contenido de la variable *length* en caso de éxito y, en caso de fallo, el número de bytes que se quedan fuera del buffer.

- Errores:

Ninguno.

- Ejemplo:

```
v3_BUFFER * buf;
u64 size, grow;
u32 data, counter, data;
if ( ( buf = v3MemAlloc( v3u64u32( sizeof( v3_BUFFER ) ),
      V3_MEM_NORMAL ) ) == NULL )
{
   /* Error in memory allocation. */
   return( V3_FAIL );
}
_V3_SET64( size, 00000000, 00000400 );
_V3_SET64( grow, 00000000, 00000200 );
if ( v3BufferCreate( buf, size, V3_BUFFER_MODE_STACK, grow, NULL,
     v3u64u32( 0 ) ) == V3_FAIL )
{
   /* Failed */
   return( V3_FAIL );
}
/* Lets insert some u32's into the buffer. They are four bytes each. */
for ( counter = 0; counter < 20; counter++ )
{
   data = counter * counter;
```

```
        if ( v3BufferPut( buf, &data, sizeof(u32) ) != V3_PASS )
                {
                    /* Error. */
                    return( V3_FAIL );
                }
        }
 /* Lets retrieve the data. We should be getting the squares of the numbers from 0 to 19
in reverse order.  */
        for ( counter = 0; counter < 20; counter ++ )
        {
          if ( v3BufferGet( &data, sizeof( u32 ), buf ) != sizeof( u32 ) )
                {
                    /* An error.*/
                    return( V3_FAIL );
                 }
          /* do something with data. */
        }
```

❋ **v3BufferUnget:**

- Sintaxis:

```
#include "buffer.h"
s32 v3BufferUnget( v3_BUFFER * buffer, const void * data, u64 length );
```

- Descripción:

  Inserta el tamaño (*length*) de los bytes de datos (*data*) dentro del buffer (*buffer*). La diferencia de la función v3BufferPut es que los datos se colocan en el orden en el que se reciben a través de la función v3BufferGet. Hay que tener cuidado con el uso de esta función cuando nos estemos moviendo en un entorno multihilo o multithreaded.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_BUFFER_ERROR_MEMORY: No hay suficiente memoria para realizar la operación.

  V3_BUFFER_ERROR_FULL: El buffer se encuentra lleno. Esta posibilidad se da si el parámetro *grow*, a la hora de crear el buffer con la función v3BufferCreate, vale 0.

  V3_BUFFER_ERROR_MODE: Modo desconocido.

- Ejemplo:

```
v3_BUFFER * buf;
u64 size, grow;
u32 data, counter, data;
if ( ( buf = v3MemAlloc( v3u64u32( sizeof( v3_BUFFER ) ),
      V3_MEM_NORMAL ) ) == NULL )
{
   /* Error in memory allocation. */
   return( V3_FAIL );
}
_V3_SET64( size, 00000000, 00000400 );
_V3_SET64( grow, 00000000, 00000200 );
if ( v3BufferCreate( buf, size, V3_BUFFER_MODE_STACK, grow, NULL,
     v3u64u32( 0 ) ) == V3_FAIL )
{
   /* Failed */
   return( V3_FAIL );
}
/* Lets insert some u32's into the buffer. They are four bytes each. */
for ( counter = 0; counter < 20; counter++ )
{
   data = counter * counter;
   if ( v3BufferPut( buf, &data, sizeof( u32 ) ) != V3_PASS )
            {
                 /* Error. */
                 return( V3_FAIL );
            }
}
/* Lets retrieve the data. */
for ( counter = 0; counter < 20; counter ++ )
{
   if ( v3BufferGet( &data, sizeof( u32 ), buf ) != sizeof( u32 ) )
            {
                 /* An error.*/
                 return( V3_FAIL );
            }
  /* We keep pushing the data back on the stack, this means we will be getting the
same data each time */
    if ( v3BufferUnget( buf, &data, sizeof( u32 ) ) != V3_PASS )
{
     /* An error. */
     return( V3_FAIL );
}
}
```

❋ **v3BufferFree:**

- Sintaxis:

```
#include "buffer.h"
void v3BufferFree( v3_BUFFER * buffer);
```

- Descripción:

Libera todos los datos del buffer (*buffer*) y lo devuelve en un estado de no-inicialización.

- Valores devueltos:

Ninguno.

- Errores:

Ninguno.

- Ejemplo:

```
v3_BUFFER * buf;
/* Use of the buffer...*/
v3BufferFree( buf );
```

3.6.4.4 Funciones para la compresión de datos:

❋ **v3CompressInit:**

- Sintaxis:

```
#include "compress.h"
s32 v3CompressInit( v3_BUFFER * output, v3_ENCODER_TMP * tmp,
u32 compression_type, u32 direction, u32 level );
```

- Descripción:

Inicializa la salida (*output*) del buffer y, temporalmente, coloca los datos necesarios para realizar la compresión especificada en la variable *type*. La variable *level* es un número que va desde el 1 al 9 y representa el modo de compresión de datos a intentar, siendo 1 rápido y 9 intensivo.

- Tipo de compresión:

V3_COMPRESS_BZIP2: Se lleva a cabo una compresión del tipo bzip2.

V3_COMPRESS_BASE64: Se hace una codificación en base64, expandiendo los datos al 33%.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_ENCODER_ERROR_PARAM: Parámetro inválido.

  V3_ENCODER_ERROR_STATE: Estado inválido o llamada a la función en orden incorrecto.

  V3_ENCODER_ERROR_MEMORY: Problema por escasez de memoria.

  V3_ENCODER_ERROR_FATAL: Error fatal en el algoritmo.

  V3_ENCODER_ERROR_MODE: Modo inválido o los parámetros enviados al algoritmo inválidos.

- Ejemplo:

```
v3_BUFFER buf;
v3_ENCODER_TMP tmp;
v3MemSet( &buf, v3u64u32( sizeof( v3_BUFFER ) ), 0 );
v3MemSet( &tmp, v3u64u32( sizeof( v3_ENCODER_TMP ) ), 0 );
if ( v3CompressInit( &buf, &tmp, V3_COMPRESS_BZIP2,
     V3_ENCODER_ENCODE, 9 ) != V3_PASS )
{
   return( -1 );
}
```

❋ **v3Compress:**

- Sintaxis:

```
#include "compress.h"
s32 v3Compress( v3_BUFFER * output, v3_ENCODER_TMP * tmp,
const void * const data, u64 length );
```

- Descripción:

  Mantiene el tamaño (*length*) de los datos (*data*) en las rutinas de compresión. Cualquier dato comprimido se coloca dentro de la salida (*output*) del buffer pero puede ser que, después de infinidad de llamadas a diferentes algoritmos, esto no se produzca. Por esta razón, se aconseja que los datos, antes de ser comprimidos, sean salvados a través de la función v3Save.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_ENCODER_ERROR_PARAM: Parámetro inválido.

  V3_ENCODER_ERROR_STATE: Estado inválido o llamada a la función en orden incorrecto.

  V3_ENCODER_ERROR_MEMORY: Problema por escasez de memoria.

  V3_ENCODER_ERROR_FATAL: Error fatal en el algoritmo.

- Ejemplo:

```
v3_BUFFER buf;
v3_ENCODER_TMP tmp;
u8 data[8] = { 0, 1, 2, 3, 3, 3, 3, 4 };
u32 i;
/* ... */
for ( i = 0 ; i < 100 ; i++ )
{
   if ( v3Compress( &buf, &tmp, data, v3u64u32( 8 ) ) != V3_PASS )
            {
                return( -2 );
            }
}
```

❋ **v3CompressEnd:**

- Sintaxis:

```
#include "compress.h"
s32 v3CompressEnd( v3_BUFFER * output, v3_ENCODER_TMP * tmp );
```

- Descripción:

  Limpia cualquier tipo de dato temporal y lo coloca al final de los datos comprimidos que se encuentran dentro del buffer. Hay que asegurarse de vaciar la salida del buffer (v3BufferFree) después de realizar una llamada a esta función.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_ENCODER_ERROR_PARAM: Parámetro inválido.

  V3_ENCODER_ERROR_STATE: Estado inválido o llamada a la función en orden incorrecto.

  V3_ENCODER_ERROR_FATAL: Error fatal en el algoritmo.

- Ejemplo:

```
v3_BUFFER buf;
v3_ENCODER_TMP tmp;
/* ... */
if ( v3CompressEnd( &buf, &tmp ) != V3_PASS )
{
   /* v3CompressEnd() must be called again on an error */
   v3CompressEnd( &tmp );
   return( -3 );
}
/* empty out the buffer */
/* then dont forget to free it */
v3BufferFree( &buf );
```

❋ **v3DecompressInit:**

- Sintaxis:

```
#include "compress.h"
s32 v3DecompressInit( v3_BUFFER * output, v3_ENCODER_TMP * tmp,
u32 compression_type, u32 direction, u32 level );
```

- Descripción:

  Inicializa la salida (*output*) del buffer y, temporalmente, coloca los datos necesarios para realizar la descompresión especificada en la variable *type*. La variable *level* es un número que va desde el 1 al 9 y representa el modo de descompresión de datos a intentar, siendo 1 rápido y 9 intensivo.

- Tipo de descompresión:

  V3_DECOMPRESS_BZIP2: Se lleva a cabo una descompresión del tipo bzip2.

  V3_DECOMPRESS_BASE64: Se hace una descodificación en base64, expandiendo los datos al 33%.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_ENCODER_ERROR_PARAM: Parámetro inválido.

  V3_ENCODER_ERROR_STATE: Estado inválido o llamada a la función en orden incorrecto.

  V3_ENCODER_ERROR_MEMORY: Problema por escasez de memoria.

  V3_ENCODER_ERROR_FATAL: Error fatal en el algoritmo.

  V3_ENCODER_ERROR_MODE: Modo inválido o los parámetros enviados al algoritmo inválidos.

- Ejemplo:

```
v3_BUFFER buf;
v3_ENCODER_TMP tmp;
v3MemSet( &buf, v3u64u32( sizeof( v3_BUFFER ) ), 0 );
v3MemSet( &tmp, v3u64u32( sizeof( v3_ENCODER_TMP ) ), 0 );
if ( v3CompressInit( &buf, &tmp, V3_COMPRESS_BZIP2,
     V3_ENCODER_ENCODE, 9 ) != V3_PASS )
{
   return( -1 );
}
```

❋ **v3Decompress:**

- Sintaxis:

```
#include "compress.h"
s32 v3Decompress( v3_BUFFER * output, v3_ENCODER_TMP * tmp,
const void * const data, u64 length );
```

- Descripción:

  Mantiene el tamaño (*length*) de los datos (*data*) en las rutinas de descompresión. Cualquier dato comprimido se coloca dentro de la salida (*output*) del buffer pero puede ser que, después de infinidad de llamadas a diferentes algoritmos, esto no se produzca. Por esta razón, se aconseja que los datos, antes de ser comprimidos, sean salvados a través de la función v3Save.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_ENCODER_ERROR_PARAM: Parámetro inválido.

  V3_ENCODER_ERROR_STATE: Estado inválido o llamada a la función en orden incorrecto.

  V3_ENCODER_ERROR_MEMORY: Problema por escasez de memoria.

  V3_ENCODER_ERROR_FATAL: Error fatal en el algoritmo.

- Ejemplo:

```
v3_BUFFER buf;
v3_ENCODER_TMP tmp;
u8 data[8] = { 0, 1, 2, 3, 3, 3, 3, 4 };
u32 i;
/* ... */
for ( i = 0 ; i < 100 ; i++ )
{
   if ( v3Decompress( &buf, &tmp, data, v3u64u32( 8 ) ) != V3_PASS )
            {
                return( -2 );
            }
}
```

❋ **v3DecompressEnd:**

- Sintaxis:

```
#include "compress.h"
s32 v3DecompressEnd( v3_BUFFER * output, v3_ENCODER_TMP * tmp );
```

- Descripción:

  Limpia cualquier tipo de dato temporal y lo coloca al final de los datos descomprimidos que se encuentran dentro del buffer. Hay que asegurarse de vaciar la salida del buffer (v3BufferFree) después de realizar una llamada a esta función.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_ENCODER_ERROR_PARAM: Parámetro inválido.

  V3_ENCODER_ERROR_STATE: Estado inválido o llamada a la función en orden incorrecto.

  V3_ENCODER_ERROR_FATAL: Error fatal en el algoritmo.

- Ejemplo:

```
v3_BUFFER buf;
v3_ENCODER_TMP tmp;
/* ... */
if ( v3DecompressEnd( &buf, &tmp ) != V3_PASS )
{
   /* v3DecompressEnd() must be called again on an error */
   v3DecompressEnd( &tmp );
   return( -3 );
}
/* empty out the buffer */
/* then dont forget to free it */
v3BufferFree( &buf );
```

3.6.4.5 Funciones para la configuración de los ficheros del programa:

❋ **v3ConfigLoad:**

- Sintaxis:

```
#include "config.h"
s32 v3ConfigLoad( v3_CONFIG * config, const ascii * filename );
```

- Descripción:

  Intenta abrir el fichero cuyo nombre viene establecido en la variable *filename* e intenta leer la configuración de los datos. Para saber cómo se tienen que especificar el path y el nombre del fichero, se aconseja mirar en la librería cosmfile.h la función v3FileOpen. Si la variable *filename* se encuentra a NULL, se creará una configuración vacía.

La configuración de un fichero tendrá el siguiente aspecto:

<begin file>
[section1]\n
key1=value1\n
key2=value2\n
\n
[section2]\n
key1=value1\n
key2=value2\n
\n
</end file>

Para la configuración de los ficheros existen los siguientes criterios:

- Los ficheros son ficheros binarios y no de texto, puesto que se supone que éstos no han de ser editados.
- La configuración viene establecida en ASCII, no en UNICODE.
- No se usan "=" ó "[" en los apartados de *keys* y se evitan los "]".
- En todos los casos se da la sensibilidad a las mayúsculas (CASE SENSITIVE).
- Las líneas en blanco se ignoran.
- Cualquier formato de línea impropio es descartado.
- Las líneas de comentario no se contemplan.
- En la mayoría de los casos, la configuración es de sólo lectura aunque se puede optimizar para ser de escritura.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error (V3_FILE_ERROR_?) si se ha producido un error.

- Errores:

  Ninguno.

- Ejemplo:

```
v3_CONFIG * config;
s32 result;
config = v3MemAlloc( v3u64u32( sizeof(v3_CONFIG) ),
V3_MEM_NORMAL );
result = v3ConfigLoad( config, (ascii *) "program.cfg" );
if ( result != V3_PASS )
{
   /* error */
}
```

❋ **v3ConfigSave:**

- Sintaxis:

```
#include "config.h"
s32 v3ConfigSave( const v3_CONFIG * config, const ascii * filename );
```

- Descripción:

Salva la configuración de los datos del fichero especificado en la variable *filename*. Para saber cómo se tienen que especificar el path y el nombre del fichero, se aconseja mirar en la librería cosmfile.h la función v3FileOpen. Para saber algo más sobre la configuración, mirar la función v3ConfigLoad.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error (V3_FILE_ERROR_?) si se ha producido un error.

- Errores:

Ninguno.

- Ejemplo:

```
v3_CONFIG * config;
s32 result;
config = v3MemAlloc( v3u64u32( sizeof(v3_CONFIG) ),
V3_MEM_NORMAL );
result = v3ConfigLoad( config, (ascii *) "program.cfg" );
if ( result != V3_PASS )
{
   /* error */
}
/* ... */
result = v3ConfigSave( config, (ascii *) "program.cfg" );
if ( result != V3_PASS )
{
   /* error */
}
```

❋ **v3ConfigSet:**

- Sintaxis:

```
#include "config.h"
s32 v3ConfigSet( v3_CONFIG * config, const ascii * section,
const ascii * key, const ascii * value );
```

- Descripción:

Establece el valor incluido en la variable *value* para el de las variables *section/key*.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

Ninguno.

- Ejemplo:

```
v3_CONFIG * config;
s32 result;
config = v3MemAlloc( v3u64u32( sizeof(v3_CONFIG) ),
V3_MEM_NORMAL );
result = v3ConfigLoad( config, (ascii *) "program.cfg" )
if ( result != V3_PASS )
{
   /* error */
}
result = v3ConfigSet( config, (ascii *) "Options",(ascii *) "logging", (ascii *)
"1" );
if ( result != V3_PASS )
{
   /* error */
}
```

❋ **v3ConfigGet:**

- Sintaxis:

```
#include "config.h"
const ascii * v3ConfigGet( v3_CONFIG * config, const ascii * section,
const ascii * key );
```

- Descripción:

Obtiene el valor del par de valores *section/key*.

- Valores devueltos:

La función nos devolverá un puntero para el valor en caso de éxito y NULL en caso de error.

- Errores:

Ninguno.

- Ejemplo:

```
v3_CONFIG * config;
s32 result;
ascii * value;
config = v3MemAlloc( v3u64u32( sizeof(v3_CONFIG) ),
V3_MEM_NORMAL );
result = v3ConfigLoad( config, (ascii *) "program.cfg" );
if ( result != V3_PASS )
{
   /* error */
}
value = v3ConfigGet( config, (ascii *) "Program",(ascii *) "interval" );
```

❋ **v3ConfigFree:**

- Sintaxis:

```
#include "config.h"
void v3ConfigFree( v3_CONFIG * config );
```

- Descripción:

Libera la configuración interna de los datos.

- Valores devueltos:

Ninguno.

- Errores:

Ninguno.

- Ejemplo:

```
v3_CONFIG * config;
config = v3MemAlloc( v3u64u32( sizeof(v3_CONFIG) ),
V3_MEM_NORMAL );
/* use the config */
v3ConfigFree( config );
v3MemFree( config );
```

3.6.4.6 Funciones de ficheros y directorios:

❋ **v3FileOpen:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3FileOpen( v3_FILE * file, const ascii * filename,
u32 mode, u32 lock );
```

- Descripción:

Intenta abrir el fichero cuyo nombre se encuentra la variable *filename* bajo un modo de fichero (*mode*) y con un tipo de bloqueo (*lock*).
Los path siempre tienen el mismo formato "*native/path/path2/filename*". El "*other slash*" se traduce en el momento, pero internamente se almacena en el mismo formato. La parte del path que refleja *native* consiste en reflejar las diferentes unidades, nodos existentes en el Sistema operativo (por ejemplo: "e:|/temp/a.txt" para win32). Si el path es enviado fuera de la red, esta parte de la que estamos hablando, *native*, puede ser quitada.

- Modos de Fichero:

  V3_FILE_MODE_EXIST: Comprueba la existencia del fichero.

  V3_FILE_MODE_READ: El fichero es abierto para lectura.

  V3_FILE_MODE_WRITE: El fichero es abierto para escritura.

  V3_FILE_MODE_APPEND: El fichero se abre para escritura pero colocando el puntero al final del fichero para no machacar la información existente.

  V3_FILE_MODE_CREATE: Crea el fichero si no existe previamente.

  V3_FILE_MODE_TRUNCATE: Trunca un fichero existente a la hora de abrirlo.

  V3_FILE_MODE_SYNC: Realiza funciones de sincronización (sync()) antes de volver.

- Bloqueos de Fichero:

  V3_FILE_LOCK_NONE: No es necesario ningún tipo de bloqueo, aunque se debe estar al tanto de la llegada de alguno.

  V3_FILE_LOCK_READ: Bloquea el fichero con un bloqueo de lectura. Esto quiere decir que otro proceso puede leer el fichero e incluso añadirle otros bloqueos de lectura al fichero, pero nunca se podrá abrir el fichero para escritura.

  V3_FILE_LOCK_WRITE: Bloquea el fichero con un bloqueo de escritura. No se permite el acceso, ni para lectura ni para escritura, de otros procesos a este fichero cuando se da la existencia de este tipo de bloqueo.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_FILE_ERROR_DENIED: Acceso denegado.

  V3_FILE_ERROR_NOTFOUND: Fichero o manejador no encontrado.

  V3_FILE_ERROR_WRITEMODE: Acceso para escritura denegado.

  V3_FILE_ERROR_CLOSED: El fichero está cerrado.

  V3_FILE_ERROR_NAME: Nombre de fichero incorrecto.

V3_FILE_ERROR_MODE: Conflicto en el modo de apertura.

- Ejemplo:

```
v3_FILE * file;
s32 result;
file = v3MemAlloc( v3u64u32( sizeof(v3_FILE) ), V3_MEM_NORMAL );
result = v3FileOpen( file, "/etc/fstab", V3_FILE_MODE_READ,
                          V3_FILE_LOCK_NONE );
if ( result == V3_PASS )
   v3PrintA( (ascii *) "/etc/fstab open for reading.\n" );
else
   v3PrintA( (ascii *) "Unable to open /etc/fstab for reading.\n" );
```

❋ **v3FileRead:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3FileRead( void * buffer, u64 * bytes_read, v3_FILE * file,
u64 length );
```

- Descripción:

Lee el número de bytes establecidos en la variable *length* del fichero especificado en la variable *filename* dentro del buffer (*buffer*).

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_FILE_ERROR_DENIED: Acceso denegado.

V3_FILE_ERROR_NOTFOUND: Fichero o manejador no encontrado.

V3_FILE_ERROR_CLOSED: El fichero está cerrado.

- Ejemplo:

```
v3_FILE * file;
char getbuff[31];
u64 bytesread;
s32 error;
/* ... */
error = v3FileRead( getbuff, &bytesread, file, v3u64u32( 30 ) );
if ( error != V3_PASS )
{
   v3PrintA( (ascii *) "We weren't able to read from the file.\n" );
}
else
{
   *( (u8 *) v3MemOffset( getbuff, bytesread ) ) = 0;
   v3PrintA( (ascii *) "We read %u bytes: %.30s\n",bytesread, getbuff );
}
```

❋ **v3FileWrite:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3FileWrite( v3_FILE * file, u64 * bytes_written,
const void * const buffer, u64 length );
```

- Descripción:

Escribe el número de bytes establecidos en la variable *length* del fichero especificado en la variable *filename* dentro del buffer (*buffer*). La variable *bytes_written* nos indica el número de bytes que se han escrito actualmente.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_FILE_ERROR_DENIED: Acceso denegado.

V3_FILE_ERROR_NOTFOUND: Fichero o manejador no encontrado.

V3_FILE_ERROR_CLOSED: El fichero está cerrado.

- Ejemplo:

```
ascii to_write[30];
v3_FILE * file;
u64 written;
u64 length;
s32 error;
/* ... */
v3StrCopyA( to_write, "Hello World!\n", v3u64u32( 30 ) );
length = v3StrLengthA( to_write );
error = v3FileWrite( file, &written, to_write, length );
if ( v3u64Eq( length, written ) )
{
   v3PrintA( (ascii *) "Success! %v bytes written.\n", written );
}
else
{
   /* do stuff with error */
   /* ... */
   v3PrintA( (ascii *) "Wrote only %v bytes (of %v)\n", written,length );
}
```

❋ **v3FileSeek:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3FileSeek( v3_FILE * file, u64 offset );
```

- Descripción:

Se desplaza dentro del fichero actual (en bytes). Si el fichero ha sido abierto en modo V3_FILE_MODE_APPEND, esta función no puede utilizarse.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_FILE_ERROR_CLOSED: El fichero está cerrado.

V3_FILE_ERROR_SEEK: Error en la búsqueda.

V3_FILE_ERROR_DENIED: Acceso denegado.

V3_FILE_ERROR_NOTFOUND: Fichero o manejador no encontrado.

- Ejemplo:

```
v3_FILE * file;
s32 result;
/* ... */
result = v3FileSeek( file, v3u64u32( 4096 ) );
if ( result == V3_PASS )
   v3PrintA( (ascii *) "Now at byte position 4096 in the file.\n" );
else
   v3PrintA( (ascii *) "Seek failed.\n" );
```

❋ **v3FileTell:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3FileTell( u64 * offset, v3_FILE * file );
```

- Descripción:

Obtiene el desplazamiento del fichero actual que se le pasa como parámetro.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_FILE_ERROR_CLOSED: El fichero está cerrado.

V3_FILE_ERROR_NOTFOUND: Fichero o manejador no encontrado.

V3_FILE_ERROR_TELL: Error al decir el desplazamiento.

- Ejemplo:

```
v3_FILE * file;
u64 offset;
s32 result;
/* ... */
result = v3FileTell( &offset, file );
if ( result == V3_PASS )
   v3PrintA( (ascii *) "Currently at byte offset %v in the file.\n",offset );
else
   v3PrintA( (ascii *) "Unable to read position in file.\n" );
```

❋ **v3FileEOF:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3FileEOF( v3_FILE * file );
```

- Descripción:

Comprueba si nos encontramos en el final del fichero (End Of File “EOF”).

- Valores devueltos:

La función nos devolverá V3_PASS si hay más por leer dentro del fichero, es decir, que no nos encontramos en el final del fichero, o nos devolverá V3_FILE_ERROR_EOF para indicarnos que nos encontramos en el final del fichero o un código de error en caso de fallo.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_FILE_ERROR_EOF: Final de fichero.

V3_FILE_ERROR_CLOSED: El fichero está cerrado.

V3_FILE_ERROR_TELL: Error al decir el desplazamiento.

V3_FILE_ERROR_LENGTH: Imposible descubrir el tamaño del fichero.

- Ejemplo:

```
v3_FILE * file;
/* ... */
if ( !v3FileEOF( file ) )
   v3PrintA( (ascii *) "There is more data to read.\n" );
else
            {
               v3PrintA( (ascii *) "No more data to be read.\n" );
               return( 0 );
            }
```

❋ **v3FileLength:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3FileLength( u64 * length, v3_FILE * file );
```

- Descripción:

Establece la longitud (*length*) del fichero (*file*) abierto.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_FILE_ERROR_CLOSED: El fichero está cerrado.

V3_FILE_ERROR_LENGTH: Imposible descubrir el tamaño del fichero.

- Ejemplo:

```
v3_FILE * file;
u64 length;
s32 result;
/* ... */
result = v3FileLength( &length, file );
if ( result == V3_PASS )
   v3PrintA( (ascii *) "The length of the file is %v bytes.\n", length);
else
  if ( result == V3_FILE_ERROR_CLOSED )
    v3PrintA( (ascii *) "The file is closed. Can not read length.\n" );
 else
    v3PrintA( (ascii *) "Error. Unable to read file length.\n" );
```

❋ **v3FileTruncate:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3FileTruncate( v3_FILE * file, u64 length );
```

- Descripción:

Trunca el fichero (*file*) en el tamaño de bytes establecidos en la variable *length*. El desplazamiento se moverá al final del fichero.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_FILE_ERROR_DENIED: Acceso denegado.

V3_FILE_ERROR_NOTFOUND: Fichero o manejador no encontrado.

V3_FILE_ERROR_WRITEMODE: Acceso para escritura denegado.

V3_FILE_ERROR_CLOSED: El fichero está cerrado.

- Ejemplo:

```
v3_FILE * file;
s32 result;
/* ... */
result = v3FileTruncate( file, v3u64u32( 10000 ) );
if ( result == V3_PASS )
   v3PrintA((ascii *)"File truncated to exactly 10000 bytes in length.\n" );
else
   v3PrintA( (ascii *) "Could not truncate file.\n" );
```

❋ **v3FileClose:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3FileClose( v3_FILE * file );
```

- Descripción:

Cierra el fichero especificado en la variable *file*.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_FILE_ERROR_NOTFOUND: Fichero o manejador no encontrado.

V3_FILE_ERROR_CLOSED: El fichero está cerrado.

- Ejemplo:

```
v3_FILE * file;
/* ... */
v3FileClose( file );
v3ProcessEnd( 0 );
```

❋ **v3FileDelete:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3FileDelete( const ascii * filename );
```

- Descripción:

Intenta borrar el fichero especificado por la variable *filename*. Se aconseja ojear la función v3FileOpen para saber cómo se especifica el path del nombre del fichero. Este borrado implica una "limpieza" general de los datos del fichero.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_FILE_ERROR_DENIED: Acceso denegado.

  V3_FILE_ERROR_NOTFOUND: Fichero o manejador no encontrado.

  V3_FILE_ERROR_NAME: Nombre de fichero incorrecto.

- Ejemplo:

```
s32 result;
result = v3FileDelete( "/tmp/file" );
if ( result == V3_PASS )
   v3PrintA( "File /tmp/file was successfully removed.\n" );
else
 {
   if ( result == V3_FILE_ERROR_DENIED )
     v3PrintA( (ascii *)"Unable to delete /tmp/file: Access denied.\n" );
  else
     v3PrintA( (ascii *)"Unable to delete /tmp/file: File doesn't exist.\n" );
 }
```

❋ **v3FileInfo:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3FileInfo( v3_FILE_INFO * info, const ascii * filename );
```

- Descripción:

  Obtiene información sobre el fichero. La información viene estructurada en v3_FILE_INFO y tiene el siguiente aspecto:

```
typedef struct
{
  u64 length;
  u32 type;
  u32 rights;
  v3_time create;
  v3_time modify;
  v3_time access;
} v3_FILE_INFO;
```

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Ninguno.

❋ **v3DirOpen:**

- Sintaxis:

  ```
  #include "cosmfile.h"
  s32 v3DirOpen( v3_DIR * dir, const ascii * dirname, u32 mode );
  ```

- Descripción:

  Intenta abrir el directorio especificado en la variable *dirname* bajo un modo especificado. Para saber más acerca de cómo se nombra el path, consultar la función v3FileOpen.

- Modos de Directorio:

  V3_DIR_MODE_EXIST: Comprueba solamente la existencia del directorio.

  V3_DIR_MODE_READ: Abre el directorio sólo en modo lectura.

  V3_DIR_MODE_CREATE: Crea el directorio si no existe previamente. Los directorios "padre" se crean si son necesarios.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_DIR_ERROR_DENIED: Acceso denegado.

  V3_DIR_ERROR_NOTFOUND: Directorio no encontrado.

  V3_DIR_ERROR_CREATE: Imposible de crear el nuevo directorio.

  V3_DIR_ERROR_MODE: Conflicto en el modo de apertura.

  V3_DIR_ERROR_CLOSED: Intento de reapertura de un directorio previamente abierto.

  V3_DIR_ERROR_NAME: Nombre de directorio incorrecto.

- Ejemplo:

```
v3_DIR * dir;
s32 result;
dir = v3MemAlloc( v3u64u32( sizeof( v3_DIR ) ), V3_MEM_NORMAL );
result = v3DirOpen( dir, "/var/", V3_DIR_MODE_READ );
if ( result == V3_PASS )
   v3PrintA( (ascii *)"Call to v3DirOpen passed. /var is open.\n" );
else
 {
  v3PrintA( (ascii *) "Call to v3DirOpen failed. Dying.\n" );
  return( 1 );
 }
```

❋ **v3DirRead:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3DirRead( v3_FILENAME * buffer, u64 * names_read, u64 length,
v3_DIR * dir );
```

- Descripción:

Lee la cantidad de bytes que le marca la variable *length* del directorio establecido por la variable *dir* dentro del buffer (*buffer*). En la variable *names_read* se almacenan los ficheros leídos. En caso de error se almacenará un 0.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_DIR_ERROR_NOTFOUND: Directorio no encontrado.

V3_DIR_ERROR_EOF: Final del directorio.

V3_DIR_ERROR_CLOSED: Intento de reapertura de un directorio previamente abierto.

V3_DIR_ERROR_NAME: Nombre de directorio incorrecto.

- Ejemplo:

```
v3_DIR * dir;
v3_FILENAME * getbuff;
u64 count;
/* ... */
while ( v3DirRead( getbuff, &count, v3u64u32( 1 ), dir ) ) == V3_PASS )
  {
    v3PrintA( (ascii *) "File: %.256s\n", getbuff );
  }
```

❋ **v3DirDelete:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3DirDelete( const ascii * dirname );
```

- Descripción:

  Intenta borrar el directorio especificado a través de la variable *dirname*. Esta función sólo funcionará si el directorio no contiene información. Para saber especificar correctamente el path, consultar la función v3FileOpen.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_DIR_ERROR_DENIED: Acceso denegado.

  V3_DIR_ERROR_NOTFOUND: Directorio no encontrado.

  V3_DIR_ERROR_NOTEMPTY: Comprueba si el directorio se encuentra vacío; si no es así, no se ejecutará esta función.

  V3_DIR_ERROR_NAME: Nombre de directorio incorrecto.

- Ejemplo:

```
s32 result;
result = v3DirDelete( "/tmp/dir" );
if ( result == V3_PASS )
   v3PrintA( (ascii *) "/tmp/dir was successfully deleted.\n" );
else
{
   if ( result == V3_DIR_ERROR_NOTFOUND )
     v3PrintA( (ascii *) "/tmp/dir doesn't exist.\n" );
  else
     v3PrintA( (ascii *) "/tmp/dir cannot be deleted.\n" );
}
```

❋ **v3DirClose:**

- Sintaxis:

```
#include "cosmfile.h"
s32 v3DirClose( v3_DIR * dir );
```

- Descripción:

Cierra el directorio.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_DIR_ERROR_NOTFOUND: Directorio no encontrado.

V3_DIR_ERROR_CLOSED: Intento de reapertura de un directorio previamente abierto.

- Ejemplo:

```
v3_DIR * dir;
s32 result;
/* ... */
result = v3DirClose( dir );
if ( result == V3_PASS )
   v3PrintA( (ascii *) "The directory was closed.\n" );
else
   v3PrintA( (ascii *)"The directory was not open or doesn't exist.\n" );
```

❋ **v3Load:**

- Sintaxis:

```
#include "cosmfile.h"
void v3u16Load( u16 * num, const u8 * bytes );
void v3u32Load( u32 * num, const u8 * bytes );
void v3u64Load( u64 * num, const u8 * bytes );
void v3u128Load( u128 * num, const u8 * bytes );
```

- Descripción:

  Lee el número de bytes especificados a través de la variable *bytes* dentro de la parte "native" del path indicada por la variable *num*.

- Valores devueltos:

  Ninguno.

- Errores:

  Ninguno, aunque hay que decir que si la variable *byte* o la variable *num* se encuentran a NULL, no se llevará a cabo ninguna operación con esta función.

- Ejemplo:

```
u8 * bytes;
u128 num;
bytes = v3MemAlloc( v3u64u32( 8 ), V3_MEM_NORMAL );
/* ...'bytes' gets read from a file or from the net... */
v3u128Load( &num, bytes );
```

❋ **v3Save:**

- Sintaxis:

```
#include "cosmfile.h"
void v3u16Save( u16 * num, const u8 * bytes );
void v3u32Save( u32 * num, const u8 * bytes );
void v3u64Save( u64 * num, const u8 * bytes );
void v3u128Save( u128 * num, const u8 * bytes );
```

- Descripción:

  Guarda el número de bytes especificados a través de la variable *bytes* dentro de la parte "native" del path indicada por la variable *num*.

- Valores devueltos:

  Ninguno.

- Errores:

  Ninguno, aunque hay que decir que si la variable *byte* o la variable *num* se encuentran a NULL, no se llevará a cabo ninguna operación con esta función.

- Ejemplo:

```
u8 * bytes;
u128 num;
bytes = v3MemAlloc( v3u64u32(8), V3_MEM_NORMAL );
/* ... */
v3u128Save( bytes, &num );
/* 'bytes' is now safe to send to a file or over the net */
```

3.6.4.7 Función para E-Mail:

❋ **v3EmailSMTP:**

- Sintaxis:

```
#include "email.h"
s32 v3EmailSMTP( const u32 smtp_server, const u16 port,
const ascii * to, const ascii * from, const ascii * subject,
const ascii * message, u32 length );
```

- Descripción:

  Envía un mensaje establecido en la variable *message* bajo el protocolo SMTP, cuya longitud viene dada en la variable *length* y con un Título (*subject*), y usando un servidor de correo (*smtp_server*) y un puerto (*port*) dado. El destinatario viene reflejado en la variable *to* mientras que el destinatario viene almacenado en la variable *from*. SMTP es un protocolo de 7 bits pero que sólo puede mandar texto plano.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_EMAIL_ERROR_TO: La dirección destino (*to*) es rechazada por el servidor SMTP.

  V3_EMAIL_ERROR_FROM: La dirección origen (*from*) es rechazada por el servidor SMTP.

V3_EMAIL_ERROR_HOST: El servidor SMTP especificado es incorrecto.

V3_EMAIL_ERROR_MESSAGE: La longitud del mensaje es distinta de 0 mientras que el mensaje es NULL.

V3_EMAIL_ERROR_NORELAY: El servidor especificado no permite retransmisiones (relaying).

V3_EMAIL_ERROR_ABORTED: Imposible de completar la transmisión.

- Ejemplo:

```
u32 smtp_server;
ascii * server = "127.0.0.1";
ascii * to = "foo@bar.com";
ascii * from = "cosm@bar.com";
ascii * subject = "Testing";
ascii * message = "This is a test";
u32 length;
s32 error;
length = v3u32u64( v3StrLengthA( message ) );
if ( v3NetDNS( &smtp_server, 1, server ) == 1 )
{
  error = v3EmailSMTP( smtp_server, 25, to, from, subject,message, length );
  v3PrintA( "Email%.4s sent!\n",( ( error == V3_PASS ) ? "" : " NOT" ) );
}
else
{
   v3PrintA( "DNS lookup error!\n" );
}
```

3.6.4.8 Funciones para la entrada y salida de cadenas de caracteres:

❋ **v3Input:**

- Sintaxis:

```
#include "cosmio.h"
u64 v3Input( u8 * buffer, u64 length, u32 echo, u32 wait );
```

- Descripción:

Lee un número de bytes establecidos por la variable *length* de la entrada estándar (stdin) dentro del buffer (*buffer*). Si la variable *echo* vale V3_IO_NOECHO, eso quiere decir que los datos no serán “reflejados” de ninguna manera (para la introducción de passwords, etc.). En caso contrario, la variable debería valer V3_IO_ECHO. Si la variable *wait* vale V3_IO_WAIT, entonces no se devuelve nada hasta que la variable *length* se encuentre habilitada.

- Valores devueltos:

  La función nos devolverá el número de bytes leídos. Para indicarnos que nos encontramos en el final del fichero (EOF) nos devolverá 0xFFFFFFFFFFFFFFFF (-1).

- Errores:

  Ninguno.

- Ejemplo:

```
u8 * buffer;
u64 size;
u64 len;
u64 eof_length;
/* ... */
_V3_SET64( size, 00000000, 00000001 );
len = v3Input( buffer, size, V3_IO_ECHO );
_V3_SET64( eof_length, FFFFFFFF, FFFFFFFF );
if ( v3u64Eq( len, eof_length ) )
{
   /* EOF */
}
```

❋ **v3InputA:**

- Sintaxis:

```
#include "cosmio.h"
u64 v3InputA( ascii * buffer, u64 max_chars, u32 echo );
```

- Descripción:

  Lee un número de bytes ASCII determinados mediante la variable *max_chars* menos 1 de la unidad de entrada estándar hasta el final del fichero (EOF) o hasta que nos encontremos "\n". El buffer (*buffer*) será un string completo si la entrada procede de una lectura. Si la variable *echo* vale V3_IO_NOECHO, eso quiere decir que los datos no serán "reflejados" de ninguna manera (para la introducción de passwords, etc.). En caso contrario, la variable debería valer V3_IO_ECHO. Cualquier entrada más allá de *max_chars*-1 deberá ser descartada para que no se nos desborde el buffer.

- Valores devueltos:

  La función nos devolverá el número de bytes leídos. Para indicarnos que nos encontramos en el final del fichero (EOF) nos devolverá 0xFFFFFFFFFFFFFFFF (-1).

- Errores:

  Ninguno.

- Ejemplo:

```
ascii * buffer;
u64 size;
u64 len;
u64 eof_length;
/* ... */
_V3_SET64( size, 00000000, 000000FF );
len = v3InputA( buffer, size, V3_IO_ECHO );
_V3_SET64( eof_length, FFFFFFFF, FFFFFFFF );
if ( v3u64Eq( len, eof_length ) )
{
   /* EOF */
}
```

  This will read up to *size*-1 characters into *buffer* from standard input.

❋ **v3InputU:**

- Sintaxis:

```
#include "cosmio.h"
u64 v3InputU( unicode * buffer, u64 max_chars, u32 echo );
```

- Descripción:

  Lee un número de bytes UNICODE determinados mediante la variable *max_chars* menos 1 de la unidad de entrada estándar hasta el final del fichero (EOF) o hasta que nos encontremos "\n". El buffer (*buffer*) será un string completo si la entrada procede de una lectura. Si la variable *echo* vale V3_IO_NOECHO, eso quiere decir que los datos no serán "reflejados" de ninguna manera (para la introducción de passwords, etc.). En caso contrario, la variable debería valer V3_IO_ECHO.

- Valores devueltos:

  La función nos devolverá el número de bytes leídos. Para indicarnos que nos encontramos en el final del fichero (EOF) nos devolverá 0xFFFFFFFFFFFFFFFF (-1).

- Errores:

  Ninguno.

- Ejemplo:

```
unicode * buffer;
u64 size;
u64 len;
u64 eof_length;
/* ... */
_V3_SET64( size, 00000000, 000000FF );
len = v3InputU( buffer, size, V3_IO_ECHO );
_V3_SET64( eof_length, FFFFFFFF, FFFFFFFF );
if ( v3u64Eq( len, eof_length ) )
  {
    /* EOF */
}
```

This will read up to *size*-1 characters into *buffer* from standard input.

❋ **v3StrLengthA:**

- Sintaxis:

```
#include "cosmio.h"
u64 v3StrLengthA( const ascii * string );
```

- Descripción:

Obtiene la longitud de la cadena ASCII señalada por el puntero *string*.

- Valores devueltos:

La función nos devolverá el tamaño de la cadena ASCII en número de caracteres.

- Errores:

Ninguno.

- Ejemplo:

```
ascii * string;
u64 size;
u64 length;
_V3_SET64( size, 00000000, 00000400 );
string = v3MemAlloc( size, (u32) 0 );
string = (ascii *) "A 27 character test string.";
length = v3StrLengthA( string );
```

❋ **v3StrLengthU:**

- Sintaxis:

```
#include "cosmio.h"
u64 v3StrLengthU( const unicode * string );
```

- Descripción:

Obtiene la longitud de la cadena UNICODE señalada por el puntero *string*.

- Valores devueltos:

La función nos devolverá el tamaño de la cadena UNICODE en número de caracteres.

- Errores:

Ninguno.

- Ejemplo:

```
unicode string[256];
u64 size;
u64 length;
_V3_SET64( size, 00000000, 00000100 );
if ( v3StrCopyUA( string,(ascii *) "A 27 character test string.", size ) !=
V3_PASS )
{
   /* Error */
}
length = v3StrLengthU( string );
```

❋ **v3StrCopyA:**

- Sintaxis:

```
#include "cosmio.h"
s32 v3StrCopyA( ascii * dest, const ascii * src, u64 max_chars );
```

- Descripción:

Copia un número de caracteres ASCII establecidos en la variable *max_chars* menos 1 de la cadena origen establecida en la variable *src* a la cadena destino establecida en la variable *dest*, incluido el carácter de final de string "\0". Si la cadena de origen (*src*) tiene una longitud mayor de *max_chars*-1, la función dará error y no realizará ningún tipo de operación.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Las posibles causas de fallo son las siguientes:
  - Algún parámetro era NULL ó 0.
  - La cadena de origen (*src*) era más larga de *max_chars*-1.

- Ejemplo:

```
ascii string[256];
ascii copy[256];
u64 size;
/* ... */
if ( v3StrCopyA( copy, string, size ) != V3_PASS )
  {
    /* Error */
  }
```

❋ **v3StrCopyU:**

- Sintaxis:

```
#include "cosmio.h"
s32 v3StrCopyU( unicode * dest, const unicode * src, u64 max_chars );
```

- Descripción:

  Copia un número de caracteres UNICODE establecidos en la variable *max_chars* menos 1 de la cadena origen establecida en la variable *src* a la cadena destino establecida en la variable *dest*, incluido el carácter de final de string "\0". Si la cadena de origen (*src*) tiene una longitud mayor de *max_chars*-1, la función dará error y no realizará ningún tipo de operación.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Las posibles causas de fallo son las siguientes:
  - Algún parámetro era NULL ó 0.
  - La cadena de origen (*src*) era más larga de *max_chars*-1.

- Ejemplo:

```
unicode string[256];
unicode copy[256];
u64 size;
/* ... */
if ( v3StrCopyU( copy, string, size ) != V3_PASS )
{
   /* Error */
}
```

❋ **v3StrCopyUA:**

- Sintaxis:

```
#include "cosmio.h"
s32 v3StrCopyUA( unicode * dest, const ascii * src, u64 max_chars );
```

- Descripción:

Convierte una cadena de caracteres ASCII a una cadena de caracteres UNICODE incluyendo el carácter de fin "\0". Sólo aquellas cadenas que tengan una longitud superior a *max_chars*-1 no se convertirán.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

Las posibles causas de fallo son las siguientes:

- Algún parámetro era NULL ó 0.
- La cadena de origen (*src*) era más larga de *max_chars*-1.

- Ejemplo:

```
unicode buffer[256];
u64 size;
_V3_SET64( size, 00000000, 00000100 );
/* ... */
if ( v3StrCopyUA( buffer, (ascii *) "Test string", size )!= V3_PASS )
{
   /* Error */
}
```

❋ **v3StrCmpA:**

- Sintaxis:

```
#include "cosmio.h"
s32 v3StrCmpA( const ascii * stringA, const ascii * stringB, u64 max_chars );
```

- Descripción:

Compara dos cadenas de caracteres ASCII desde la posición que nos indica la variable *max_chars* hasta el final de la cadena.

- Valores devueltos:

La función nos devolverá un 0 si las dos cadenas que estamos comparando son idénticas. Nos devolverá un número positivo si el primer carácter ASCII diferente en ambas cadenas es mayor en la primera cadena (*stringA*) que en la segunda (*stringB*) y nos devolverá un número negativo en el caso contrario, es decir, que el carácter sea menor. Si cualquier parámetro de la función es NULL ó 0, entonces la función nos devolverá –1 a menos que amabas cadenas (*stringA*, *stringB*) sean idénticas.

- Errores:

Ninguno.

- Ejemplo:

```
ascii stringa[128];
ascii stringb[128];
u64 max;
/* set stringa and stringb to something */
if ( v3StrCmpA( stringa, stringb, max ) != 0 )
{
   /* Strings do not match */
}
```

❋ **v3StrCmpU:**

- Sintaxis:

```
#include "cosmio.h"
s32 v3StrCmpU( const unicode * stringA, const unicode * stringB, u64
max_chars );
```

- Descripción:

Compara dos cadenas de caracteres UNICODE desde la posición que nos indica la variable *max_chars* hasta el final de la cadena.

- Valores devueltos:

La función nos devolverá un 0 si las dos cadenas que estamos comparando son idénticas. Nos devolverá un número positivo si el primer carácter UNICODE diferente en ambas cadenas es mayor en la primera cadena (*stringA*) que en la segunda (*stringB*) y nos devolverá un número negativo en el caso contrario, es decir, que el carácter sea menor. Si cualquier parámetro de la función es NULL ó 0, entonces la función nos devolverá –1 a menos que amabas cadenas (*stringA, stringB*) sean idénticas.

- Errores:

Ninguno.

- Ejemplo:

```
unicode stringa[128];
unicode stringb[128];
u64 max;
/* set stringa and stringb to something */
if ( v3StrCmpU( stringa, stringb, max ) != 0 )
{
   /* Strings do not match */
}
```

❋ **v3StrCharA:**

- Sintaxis:

```
#include "cosmio.h"
ascii * v3StrCharA( const ascii * string, ascii match_char, u64 max_chars );
```

- Descripción:

  Localiza la primera coincidencia con el carácter ASCII que estamos buscando (*match_char*) en la cadena *string*.

- Valores devueltos:

  La función nos devolverá un puntero que nos señale dónde se encuentra esa coincidencia con el carácter ASCII que buscamos dentro de la cadena que estamos explorando o nos devolverá NULL en caso contrario.

- Errores:

  Ninguno.

- Ejemplo:

```
ascii stringa[128];
ascii * found;
u64 max;
/* read in string */
if ( ( found = v3StrCharA( stringa, 'a', max ) ) != NULL )
{
   /* the character 'a' was found in the string */
}
```

❋ **v3StrCharU:**

- Sintaxis:

```
#include "cosmio.h"
ascii * v3StrCharU( const unicode * string, unicode match_char, u64
max_chars );
```

- Descripción:

  Localiza la primera coincidencia con el carácter UNICODE que estamos buscando (*match_char*) en la cadena *string*.

- Valores devueltos:

  La función nos devolverá un puntero que nos señale dónde se encuentra esa coincidencia con el carácter UNICODE que buscamos dentro de la cadena que estamos explorando o nos devolverá NULL en caso contrario.

- Errores:

  Ninguno.

- Ejemplo:

```
unicode stringa[128];
unicode * found;
u64 max;
/* read in string */
if ( ( found = v3StrCharU( stringa, (unicode) 'a', max ) ) != NULL )
{
   /* the character 'a' was found in the string */
}
```

❋ **v3StrStrA:**

- Sintaxis:

```
#include "cosmio.h"
ascii * v3StrStrA( const ascii * string,const ascii * substring,u64 max_chars );
```

- Descripción:

  Localiza una subcadena (*substring*) de caracteres ASCII dentro de una cadena (*string*) de caracteres del mismo tipo.

- Valores devueltos:

  La función nos devolverá un puntero que nos señale dónde se encuentra la primera posición dentro de la cadena en la que hemos encontrado la subcadena que estábamos buscando, o nos devolverá NULL en caso contrario.

- Errores:

  Ninguno.

❋ **v3StrStrU:**

- Sintaxis:

```
#include "cosmio.h"
ascii * v3StrStrU( const unicode * string,const unicode * substring,u64
max_chars );
```

- Descripción:

  Localiza una subcadena (*substring*) de caracteres UNICODE dentro de una cadena (*string*) de caracteres del mismo tipo.

- Valores devueltos:

  La función nos devolverá un puntero que nos señale dónde se encuentra la primera posición dentro de la cadena en la que hemos encontrado la subcadena que estábamos buscando, o nos devolverá NULL en caso contrario.

- Errores:

  Ninguno.

❋ **v3{integral type}A:**

- Sintaxis:

```
#include "cosmio.h"
s32 v3u32A( u32 * result, ascii ** end, const ascii * string, u32 radix );
s32 v3s32A( s32 * result, ascii ** end, const ascii * string, u32 radix );
s32 v3u64A( u64 * result, ascii ** end, const ascii * string, u32 radix );
s32 v3s64A( s64 * result, ascii ** end, const ascii * string, u32 radix );
s32 v3u128A( u128 * result, ascii ** end, const ascii * string, u32 radix );
s32 v3s128A( s128 * result, ascii ** end, const ascii * string, u32 radix );
```

- Descripción:

  Convierte una cadena de caracteres ASCII (*string*) en un número almacenándolo en la variable *radix*. Los números quedarán con el siguiente formato: [space*][+|-][0|0x|0X]{0-9a-zA-Z}+ La variable *radix* debe ser un número de 2 a 36 ó bien 0. Si esta variable vale 0, los números empezarán con “0x”, o bien “0X”, y se tratarán como números en base 16. Si los números empezaran con un solo 0, se les tratarán en base 8 y el resto de los números serán tratados en base 10. Si la variable *end* es distinta de NULL, se usará el carácter que contiene para colocarlo después del último carácter utilizado en el número.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito y además pondrá el resultado en la variable *result*, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Ninguno.

❋ **v3{integral type}U:**

- Sintaxis:

```
#include "cosmio.h"
s32 v3u32U( u32 * result, unicode ** end, const unicode * string, u32 radix );
s32 v3s32U( s32 * result, unicode ** end, const unicode * string, u32 radix );
s32 v3u64U( u64 * result, unicode ** end, const unicode * string, u32 radix );
s32 v3s64U( s64 * result, unicode ** end, const unicode * string, u32 radix );
s32 v3u128U(u128 * result,unicode ** end,const unicode * string,u32 radix );
s32 v3s128U(s128 * result,unicode ** end,const unicode * string, u32 radix );
```

- Descripción:

  Convierte una cadena de caracteres UNICODE (*string*) en un número almacenándolo en la variable *radix*. Los números quedarán con el siguiente formato: [space*][+|-][0|0x|0X]{0-9a-zA-Z}+ La variable *radix* debe ser un número de 2 a 36 ó bien 0. Si esta variable vale 0, los números empezarán con "0x", o bien "0X", y se tratarán como números en base 16. Si los números empezaran con un solo 0, se les tratarán en base 8 y el resto de los números serán tratados en base 10. Si la variable *end* es distinta de NULL, se usará el carácter que contiene para colocarlo después del último carácter utilizado en el número.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito y además pondrá el resultado en la variable *result*, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Ninguno.

❋ **v3{float type}A:**

- Sintaxis:

```
#include "cosmio.h"
s32 v3f32A( f32 * result, ascii ** end, const ascii * string );
s32 v3f64A( f64 * result, ascii ** end, const ascii * string );
```

- Descripción:

  Convierte una cadena de caracteres ASCII en un número en notación de coma flotante. Los números quedarán con el siguiente formato: [space*][+|-]{{0-9}+[.{0-9}*]|.{0-9}+}[{e|E}[+|-]{0-9}+] Si la variable *end* es distinta de NULL, se usará el carácter que contiene para colocarlo después del último carácter utilizado en el número.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito y además pondrá el resultado en la variable *result*. Si el número es demasiado grande, en esta variable almacenará +/- HUGE_VAL ó 0 si el string no tuviera una cadena numérica como contenido, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Ninguno.

❋ **v3{float type}U:**

- Sintaxis:

  ```
  #include "cosmio.h"
  s32 v3f32A( f32 * result, unicode ** end, const unicode * string );
  s32 v3f64A( f64 * result, unicode ** end, const unicode * string );
  ```

- Descripción:

  Convierte una cadena de caracteres UNICODE en un número en notación de coma flotante. Los números quedarán con el siguiente formato: [space*][+|-]{{0-9}+[.{0-9}*]|.{0-9}+}[{e|E}[+|-]{0-9}+] Si la variable *end* es distinto de NULL, se usará el carácter que contiene para colocarlo después del último carácter utilizado en el número.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito y además pondrá el resultado en la variable *result*. Si el número es demasiado grande, en esta variable almacenará +/- HUGE_VAL ó 0 si el string no tuviera una cadena numérica como contenido, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Ninguno.

❋ **v3PrintA:**

- Sintaxis:

```
#include "cosmio.h"
u64 v3PrintA( const ascii * format, ... );
```

- Descripción:

Imprime el texto, en formato ASCII, en la unidad de salida estándar. Todos los buffers de salida se llenarán antes de retornar de la función.
En el siguiente esquema, se especifica la relación de campos y valores necesarios para el buen funcionamiento de esta función:

%[width][.prec]type_char

- [width]:
  - 0n: como mínimo se imprimen n caracteres, pad 0.
  - -n: como mínimo se imprimen n caracteres con justificación a la izquierda.
  - n: como mínimo se imprimen n caracteres sin ningún tipo de distinción entre mayúsculas y minúsculas.
  - *: en el siguiente argumento (u32) se especificará el ancho.
- [.prec]:
  - .n: como máximo se imprimirán n decimales.
  - .*: en el siguiente argumento (u32) se especificará la precisión.
- type_char:
  - c: carácter simple.
  - s: cadena terminada en 0.
  - b: buffer de bytes puros; el flag de precisión es la longitud en bytes.
  - f = f32 [-]dddd.dddddd
  - g = f64 [-]dddd.dddddd
  - F = f32 [-]d.dddddd[Edd]
  - G = f64 [-]d.dddddd[Edd]

- Valores devueltos:

La función nos devolverá el número de caracteres a la salida.

- Errores:

Ninguno.

- Ejemplo:

```
ascii * output;
void * foo;
u64 size;
_V3_SET64( size, 00000000, 000000FF );
output = v3MemAlloc( size, (u32) V3_MEM_NORMAL );
/* ... */
v3PrintA( (ascii *) "Error: %.*s\n", v3u32u64( size ), output );
v3PrintA( (ascii *) "Size: %04v bytes\n", size );
v3PrintA( (ascii *) "foo points to: %p\nAddress of size is: %p\n",foo, &size );
```

This would print something similar to (on a 32-bit machine):

```
Error: Timeout -- try again later
Size: 0255 bytes
foo points to: 08049860
Address of size is: 7FFFF7A8
```

❋ **v3PrintU:**

- Sintaxis:

```
#include "cosmio.h"
u64 v3PrintU( const unicode * format, ... );
```

- Descripción:

Imprime el texto, en formato UNICODE, en la unidad de salida estándar. Todos los buffers de salida se llenarán antes de retornar de la función.
En el siguiente esquema, se especifica la relación de campos y valores necesarios para el buen funcionamiento de esta función:

%[width][.prec]type_char

- [width]:
  - 0n: como mínimo se imprimen n caracteres, pad 0.
  - -n: como mínimo se imprimen n caracteres con justificación a la izquierda.
  - n: como mínimo se imprimen n caracteres sin ningún tipo de distinción entre mayúsculas y minúsculas.
  - *: en el siguiente argumento (u32) se especificará el ancho.
- [.prec]:
  - .n: como máximo se imprimirán n decimales.
  - .*: en el siguiente argumento (u32) se especificará la precisión.

- type_char:
  - c: carácter simple.
  - s: cadena terminada en 0.
  - b: buffer de bytes puros; el flag de precisión es la longitud en bytes.
  - f = f32 [-]dddd.dddddd
  - g = f64 [-]dddd.dddddd
  - F = f32 [-]d.dddddd[Edd]
  - G = f64 [-]d.dddddd[Edd]

- Valores devueltos:

  La función nos devolverá el número de caracteres a la salida.

- Errores:

  Ninguno.

- Ejemplo:

```
unicode string[256];
unicode * output;
void * foo;
u64 size;
_V3_SET64( size, 00000000, 00000100 );
output = v3MemAlloc( v3u64Lsh( size, 1 ), (u32) V3_MEM_NORMAL );
/* ... */
if ( v3StrCopyUA( string, (ascii *) "Error: %.*s\n", size )!= V3_PASS )
  {
    /* Error */
  }
v3PrintU( string, v3u32u64( size ), output );
if ( v3StCopyUA( string,(ascii *) "Size: %04v bytes\n", size ) != V3_PASS )
  {
    /* Error */
  }
v3PrintU( string, size );
if ( v3StrCopyUA( string, (ascii *)"foo points to: %p\nAddress of size is:
%p\n", size ) != V3_PASS )
  {
    /* Error */
  }
v3PrintU( string, foo, &size );
```

This would print something similar to (on a 32-bit machine):

```
Error: Timeout -- try again later
Size: 0255 bytes
foo points to: 08049860
Address of size is: 7FFFF7A8
```

❋ **v3PrintAStr:**

- Sintaxis:

```
#include "cosmio.h"
u64 v3PrintAStr( ascii * string, u64 max_chars,const ascii * format, ... );
```

- Descripción:

Imprime el texto, en formato ASCII, en una cadena de caracteres que se le pasa como parámetro a la función en la variable *string*. La diferencia que tiene con la función v3PrintA es que en ésta no se pueden imprimir cadenas que superen la longitud de *max_chars*-1.
En el siguiente esquema, se especifica la relación de campos y valores necesarios para el buen funcionamiento de esta función:

%[width][.prec]type_char

- [width]:
  - 0n: como mínimo se imprimen n caracteres, pad 0.
  - -n: como mínimo se imprimen n caracteres con justificación a la izquierda.
  - n: como mínimo se imprimen n caracteres sin ningún tipo de distinción entre mayúsculas y minúsculas.
  - *: en el siguiente argumento (u32) se especificará el ancho.
- [.prec]:
  - .n: como máximo se imprimirán n decimales.
  - .*: en el siguiente argumento (u32) se especificará la precisión.
- type_char:
  - c: carácter simple.
  - s: cadena terminada en 0.
  - b: buffer de bytes puros; el flag de precisión es la longitud en bytes.
  - f = f32 [-]dddd.dddddd
  - g = f64 [-]dddd.dddddd
  - F = f32 [-]d.dddddd[Edd]
  - G = f64 [-]d.dddddd[Edd]

- Valores devueltos:

La función nos devolverá el número de caracteres escritos ó –1 en caso de que se supere la longitud indicada por *max_chars*-1.

- Errores:

Ninguno.

- Ejemplo:

```
ascii * output;
void * foo;
u64 size;
u64 chars_output;
_V3_SET64( size, 00000000, 000000FF );
output = v3MemAlloc( size, (u32) V3_MEM_NORMAL );
/* ... */
chars_output = v3PrintAStr( output, size,(ascii *) "foo points to: %p\nAddress
of size is: %p\n",foo, &size );
if ( !v3u64Eq( chars_output, v3u64u32( 0 ) )
  {
   /* Error */
  }
```

❋ **v3PrintUStr:**

- Sintaxis:

```
#include "cosmio.h"
u64 v3PrintUStr(unicode * string,u64 max_chars,const unicode * format, ... );
```

- Descripción:

Imprime el texto, en formato UNICODE, en una cadena de caracteres que se le pasa como parámetro a la función en la variable *string*. La diferencia que tiene con la función v3PrintU es que en ésta no se pueden imprimir cadenas que superen la longitud de *max_chars*-1.
En el siguiente esquema, se especifica la relación de campos y valores necesarios para el buen funcionamiento de esta función:

%[width][.prec]type_char

- [width]:
  - 0n: como mínimo se imprimen n caracteres, pad 0.
  - -n: como mínimo se imprimen n caracteres con justificación a la izquierda.
  - n: como mínimo se imprimen n caracteres sin ningún tipo de distinción entre mayúsculas y minúsculas.
  - *: en el siguiente argumento (u32) se especificará el ancho.
- [.prec]:
  - .n: como máximo se imprimirán n decimales.
  - .*: en el siguiente argumento (u32) se especificará la precisión.

- type_char:
  - c: carácter simple.
  - s: cadena terminada en 0.
  - b: buffer de bytes puros; el flag de precisión es la longitud en bytes.
  - f = f32 [-]dddd.dddddd
  - g = f64 [-]dddd.dddddd
  - F = f32 [-]d.dddddd[Edd]
  - G = f64 [-]d.dddddd[Edd]

- Valores devueltos:

  La función nos devolverá el número de caracteres escritos ó –1 en caso de que se supere la longitud indicada por *max_chars*-1.

- Errores:

  Ninguno.

- Ejemplo:

```
unicode string[256];
unicode * output;
void * foo;
u64 size;
u64 chars_output;
_V3_SET64( size, 00000000, 00000100 );
output = v3MemAlloc( v3u64Lsh( size, 1 ), (u32) V3_MEM_NORMAL );
/* ... */
if ( v3StrCopyUA( string, (ascii *)"foo points to: %p\nAddress of size is:
%p\n", size ) != V3_PASS )
  {
   /* Error */
  }
chars_output = v3PrintUStr( output, size, string, foo, &size );
if ( v3u64Eq( chars_output, v3u64u32( 0 ) )
  {
   /* Error */
  }
```

❋ **v3PrintAFile:**

- Sintaxis:

```
#include "cosmio.h"
u64 v3PrintAFile( v3_FILE * file, const ascii * format, ... );
```

- Descripción:

Imprime el texto, en formato ASCII, en un fichero que se le especifica como parámetro a la función a través de la variable *file*.
En el siguiente esquema, se especifica la relación de campos y valores necesarios para el buen funcionamiento de esta función:

%[width][.prec]type_char

- [width]:
  - 0n: como mínimo se imprimen n caracteres, pad 0.
  - -n: como mínimo se imprimen n caracteres con justificación a la izquierda.
  - n: como mínimo se imprimen n caracteres sin ningún tipo de distinción entre mayúsculas y minúsculas.
  - *: en el siguiente argumento (u32) se especificará el ancho.
- [.prec]:
  - .n: como máximo se imprimirán n decimales.
  - .*: en el siguiente argumento (u32) se especificará la precisión.
- type_char:
  - c: carácter simple.
  - s: cadena terminada en 0.
  - b: buffer de bytes puros; el flag de precisión es la longitud en bytes.
  - f = f32 [-]dddd.dddddd
  - g = f64 [-]dddd.dddddd
  - F = f32 [-]d.dddddd[Edd]
  - G = f64 [-]d.dddddd[Edd]

- Valores devueltos:

La función nos devolverá el número de caracteres escritos en el fichero.

- Errores:

Ninguno.

- Ejemplo:

```
ascii * output;
u64 chars_written;
u64 size;
v3_FILE * file;
_V3_SET64( size, 00000000, 000000FF );
/* ... */
chars_written = v3PrintAFile( file, (ascii *) "Error: %.*s\n",
v3u32u64( size ), output );
if ( !v3u64Eq( chars_written, v3u64u32( 0 ) ) )
  {
    /* Error */
  }
```

❋ **v3PrintUFile:**

- Sintaxis:

```
#include "cosmio.h"
u64 v3PrintUFile( v3_FILE * file, const unicode * format, ... );
```

- Descripción:

Imprime el texto, en formato UNICODE, en un fichero que se le especifica como parámetro a la función a través de la variable *file*.
En el siguiente esquema, se especifica la relación de campos y valores necesarios para el buen funcionamiento de esta función:

%[width][.prec]type_char

- [width]:
  - 0n: como mínimo se imprimen n caracteres, pad 0.
  - -n: como mínimo se imprimen n caracteres con justificación a la izquierda.
  - n: como mínimo se imprimen n caracteres sin ningún tipo de distinción entre mayúsculas y minúsculas.
  - *: en el siguiente argumento (u32) se especificará el ancho.
- [.prec]:
  - .n: como máximo se imprimirán n decimales.
  - .*: en el siguiente argumento (u32) se especificará la precisión.
- type_char:
  - c: carácter simple.
  - s: cadena terminada en 0.
  - b: buffer de bytes puros; el flag de precisión es la longitud en bytes.
  - f = f32 [-]dddd.dddddd
  - g = f64 [-]dddd.dddddd
  - F = f32 [-]d.dddddd[Edd]
  - G = f64 [-]d.dddddd[Edd]

- Valores devueltos:

  La función nos devolverá el número de caracteres escritos en el fichero.

- Errores:

  Ninguno.

- Ejemplo:

```
unicode string[256];
u64 chars_written;
u64 size;
v3_FILE * file;
_V3_SET64( size, 00000000, 00000100 );
/* ... */
if ( v3StrCopyUA( string, (ascii *) "Test String.", size )!= V3_PASS )
  {
    /* Error */
  }
chars_written = v3PrintUFile( file, string );
if ( v3u64Eq( chars_written, v3u64u32( 0 ) ) )
  {
    /* Error */
  }
```

3.6.4.9 Funciones para la firma de datos, cifrado y funciones hash:

❋ **v3HashBegin:**

- Sintaxis:

```
#include "security.h"
s32 v3HashBegin( v3_HASH_TMP * tmp,
s32 (*begin)( v3_HASH_TMP * ),
s32 (*update)( v3_HASH_TMP *, const u8 *, u64 ),
s32 (*final)( v3_HASH *, v3_HASH_TMP * ) );
```

- Descripción:

  Inicializa la estructura temporal de trabajo (*work*) y configura los perfiles de los datos de “hashing”. La aplicación está diseñada para que este aspecto resulte algo trivial y poder añadir en un futuro, con gran facilidad, más diversidad de algoritmos de hash de los ya existentes en ella. En la variable *begin* se almacena un puntero que apunta a v3_HASH_TMP y devuelve un 1 en caso de éxito o un 0 en caso de error. La variable *update* también almacena un puntero a v3_HASH_TMP, pero esta vez es un puntero byte y nos devolverá los mismos valores que la variable anteriormente comentada. La variable *final* almacena un puntero que apunta al resultado de la función hash y otro para v3_HASH_TMP,

devolviendo los mismos valores que las dos anteriores. En esta aplicación los algoritmos hash que están incluidos son MD5 SHA1, pero esto puede ser ampliable como se ha comentado con anterioridad. En caso de añadir un algoritmo nuevo, también habrá que añadir una definición similar a la existente para SHA1 en v3_HASH_SHA1 para usuarios.

- Funciones hash predefinidas:

  v3_HASH_SHA1: 160 bits, predefinidas para firmas hash.

  v3_HASH_MD5: 128 bits, predefinidas para checksums.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_HASH_ERROR_PARAM: El usuario a pasado un puntero a NULL.

  V3_HASH_ERROR_MEMORY: Problemas de escasez de memoria.

- Ejemplo:

```
v3_HASH_TMP work;
v3MemSet( &work, v3u64u32( sizeof( v3_HASH_TMP ) ), 0 );
if ( v3HashBegin( &work, V3_HASH_MD5 ) != V3_PASS )
  {
    /* Failed */
    return( V3_FAIL );
  }
  /*
    or...
    if ( v3HashBegin( &work, V3_HASH_SHA1 ) != V3_PASS )
 */
```

❋ **v3Hash:**

- Sintaxis:

```
#include "security.h"
s32 v3Hash( v3_HASH_TMP * tmp, const u8 * const data, u64 length );
```

- Descripción:

  Alimenta la zona de trabajo para la función hash (*work*) de una serie de bytes de datos (*data*) cuya cantidad viene marcada por la variable *length*. Antes de usar esta función, se aconseja almacenar o salvar todos los datos; para ello hay que hacer uso de la función v3Save.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_HASH_ERROR_STATE: Estado incorrecto u orden incorrecta.

  V3_HASH_ERROR_PARAM: El usuario a pasado un puntero a NULL.

- Ejemplo:

```
ascii string[2388] = V3_TEST_TEXT_BLOCK;
v3_HASH_TMP work;
/* v3HashBegin() ... */
if ( v3Hash( &work, (ascii *) &string, v3u64u32(sizeof( string ) ) ) !=
V3_PASS )
  {
   /* Failed */
   return( V3_FAIL );
  }
```

❋ **v3HashEnd:**

- Sintaxis:

```
#include "security.h"
s32 v3HashEnd( v3_HASH * hash, v3_HASH_TMP * tmp );
```

- Descripción:

  Coge la zona de trabajo de la función hash (*work*) y genera el resultado final de la función hash y lo almacena en la variable *hash*.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_HASH_ERROR_STATE: Estado incorrecto u orden incorrecta.

  V3_HASH_ERROR_MEMORY: Problemas de escasez de memoria.

- Ejemplo:

```
v3_HASH_TMP work;
v3_HASH hash;
/* v3HashBegin() and v3Hash() ... */
if ( v3HashEnd( &hash, &work ) != V3_PASS )
  {
   /* Failed */
   return( V3_FAIL );
  }
```

❋ **v3HashEq:**

- Sintaxis:

```
#include "security.h"
s32 v3HashEq( const v3_HASH * hashA, const v3_HASH * hashB );
```

- Descripción:

  Comprueba si los resúmenes (“hashes”, *hashA, hashB*) son iguales.

- Valores devueltos:

  La función nos devolverá un 1 si los resúmenes son iguales y un 0 en caso de que no lo sean.

- Errores:

  Ninguno.

- Ejemplo:

```
v3_HASH hashA, hashB;
if( !v3HashEq( &hashA, &hashB ) )
  {
   /* Hashes are not equal */
  }
```

❋ **v3Random:**

- Sintaxis:

```
#include "security.h"
s32 v3Random( v3_RANDOM * random, u64 length, u8 * data,
u64 salt_length, const u8 * salt );
```

- Descripción:

Inicializa un generador de números aleatorios. El generador puede ser determinista o no determinista, dependiendo del valor que posea la variable *salt*. Si esta variable vale NULL, el generador será determinista. Si vale distinto de NULL, estaremos en un generador no determinista.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

Ninguno.

❋ **v3EncryptInit:**

- Sintaxis:

```
#include "security.h"
s32 v3EncryptInit( v3_BUFFER * output, v3_ENCODER_TMP * tmp,
u32 encryption_type, u32 direction, u32 mode,
const u8 * key, u32 key_bits, const u8 * iv );
```

- Descripción:

Inicializa la salida del buffer (*output*) y configura los datos temporales de la estructura de datos temporal (*tmp*) usada para poder llevar a cabo el cifrado requerido. La variable *direction* puede adquirir dos tipos de valores dependiendo de la acción que queramos llevar a cabo. Si queremos cifrar, deberá valer V3_ENCODER_ENCODE y, si queremos descifrar, deberá valer V3_ENCODER_DECODE. Todos los algoritmos usan bloques de 128 bits tanto para el modo ECB como para el modo CBC. Para ambos modos la salida generada será de 16 bits.

- Tipos predefinidos de cifrado:

V3_CRYPTO_RINJDAEL

V3_CRYPTO_AES: Advanced Encryption Standar (Rinjdael).

- Modos de cifrado:

  V3_CRYPTO_MODE_CFB: 8 bits de cifrado realimentado.

  V3_CRYPTO_MODE_CBC: cifrado por bloques encadenados para ficheros.

  V3_CRYPTO_MODE_ECB: "Electronic CodeBook" para claves y números aleatorios.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_ENCODER_ERROR_PARAM: Parámetro incorrecto.

  V3_ENCODER_ERROR_STATE: Estado incorrecto u orden de llamadas a funciones incorrecto.

  V3_ENCODER_ERROR_MEMORY: Escasez de memoria.

  V3_ENCODER_ERROR_FATAL: Error fatal en el algoritmo.

  V3_ENCODER_ERROR_MODE: Modo incorrecto o parámetros del algoritmo no válidos.

- Ejemplo:

```
v3_BUFFER buf;
v3_ENCODER_TMP tmp;
v3_HASH hash;
v3MemSet( &buf, v3u64u32( sizeof( v3_BUFFER ) ), 0 );
v3MemSet( &tmp, v3u64u32( sizeof( v3_ENCODER_TMP ) ), 0 );
/* ... setup hash ... and IV if you have one */
/* Init encryption */
if ( v3EncryptInit( &buf, &tmp, V3_CRYPTO_RIJNDAEL,
V3_ENCODER_ENCODE, V3_CRYPTO_MODE_ECB, &hash,128,NULL )
!= V3_PASS )
  {
    return( -1 );
  }
```

❋ **v3Encrypt:**

- Sintaxis:

```
#include "security.h"
s32 v3Encrypt( v3_BUFFER * output, v3_ENCODER_TMP * tmp,
const void * const data, u64 length );
```

- Descripción:

Alimenta de datos (*data*) a la rutina de cifrado con una longitud de bits estipulada en la variable *length*. Cualquier dato cifrado debe ser colocado en el buffer de salida (*output*), pero hay que tener cuidado de que no permanezca ahí o sea puesto si se realizan diversas llamadas a diferentes algoritmos. Los modos ECB y CBC siempre nos proporcionarán una salida con múltiplos de 16 bytes. Es conveniente asegurarse que los datos han sido salvados antes de hacer uso de esta función.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_ENCODER_ERROR_PARAM: Parámetro incorrecto.

V3_ENCODER_ERROR_STATE: Estado incorrecto u orden de llamadas a funciones incorrecto.

V3_ENCODER_ERROR_MEMORY: Escasez de memoria.

V3_ENCODER_ERROR_FATAL: Error fatal en el algoritmo.

- Ejemplo:

```
v3_BUFFER buf;
v3_ENCODER_TMP tmp;
u8 data[8] = { 0, 1, 2, 3, 3, 3, 3, 4 };
u32 i;
/* ... */
for ( i = 0 ; i < 100 ; i++ )
  {
    if ( v3Encrypt( &buf, &tmp, data, v3u64u32( 8 ) ) != V3_PASS )
    {
      return( -2 );
    }
  }
```

❋ **v3EncryptEnd:**

- Sintaxis:

```
#include "security.h"
s32 v3EncryptEnd( v3_BUFFER * output, v3_ENCODER_TMP * tmp );
```

- Descripción:

Limpia cualquier resto de datos temporales existentes y coloca el resto de datos cifrados que queden en el buffer. Hay que asegurarse de vaciar el buffer de datos cifrados una vez usada esta función. Para poder llevar a cabo esta misión, hay que hacer uso de la función v3BufferFree. Puede ser que esta función falle. Se aconseja hacer una segunda llamada a la función que libera el buffer y porque ésta limpiará todo vestigio que pueda quedar en el buffer.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_ENCODER_ERROR_PARAM: Parámetro incorrecto.

V3_ENCODER_ERROR_STATE: Estado incorrecto u orden de llamadas a funciones incorrecto.

V3_ENCODER_ERROR_FATAL: Error fatal en el algoritmo.

- Ejemplo:

```
v3_BUFFER buf;
v3_ENCODER_TMP tmp;
/* ... */
if ( v3EncryptEnd( &buf, &tmp ) != V3_PASS )
  {
    /* v3EncryptEnd() must be called again on an error */
    v3EncryptEnd( &tmp );
    return( -3 );
  }
/* empty out the buffer */
/* then dont forget to free it */
v3BufferFree( &buf );
```

3.6.4.10 Funciones para los ficheros log:

- **v3LogOpen:**
  - Sintaxis:

    ```
    #include "log.h"
    s32 v3LogOpen( v3_LOG * log, ascii * filename, u32 max_level, u32 mode );
    ```

  - Descripción:

    Inicializa el fichero de log estableciendo el nombre del fichero y demás parámetros necesarios indicados en las variables de la función (*file_name, max_level, mode*).

  - Modos log:

    V3_LOG_MODE_NUMBER: "Logea" cualquier mensaje que tenga un nivel igual o más bajo que el indicado en *max_level*.

    V3_LOG_MODE_BITS: "Logea" cualquier mensaje en el que coincida el nivel con un bit de la variable *max_level*.

  - Valores devueltos:

    La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

  - Errores:

    Se pueden dar las siguientes salidas como códigos de error:

    V3_LOG_ERROR_MODE: El modo especificado es incorrecto.

    V3_LOG_ERROR_NAME: El nombre del fichero especificado en la variable *filename* es incorrecto.

    V3_LOG_ERROR_INIT: La variable *log* es incorrecta.

    V3_LOG_ERROR_ACCESS: Imposible obtener privilegios de escritura en el fichero de log.

- Ejemplo:

```
v3_LOG log;
s32 result;
v3MemSet( &log, v3u64u32( sizeof( v3_LOG ) ), 0 );
result = v3LogOpen( &log, "/var/log/cosmlog", 5,
V3_LOG_MODE_NUMBER );
if ( result == V3_PASS )
 {
  v3PrintA( "Log file /var/log/cosmlog initialized.\n" );
 }
else
 {
  v3PrintA( "Unable to open /var/log/cosmlog.\n" );
 }
```

❋ **v3Log:**

- Sintaxis:

```
#include "log.h"
s32 v3Log( v3_LOG * log, u32 level, u32 echo,const ascii * format, ... );
```

- Descripción:

Escribe en el fichero de log una serie de cadenas de caracteres con un formato predefinido. Para saber más del formato, consultar la función v3PrintA. La cadena será “logeada” cuando el nivel sea aceptable (por ejemplo, el nivel 0 siempre será “logeado”). Si la variable *echo* vale V3_LOG_ECHO y el nivel es aceptable, además de “logear” la cadena se pasará a imprimirla en la unidad de salida estándar. Si alguna de las dos premisas anteriores no se diera, la cadena no se imprimiría.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_LOG_ERROR_INIT: El fichero de log no se ha inicializado utilizando la función v3LogOpen.

V3_LOG_ERROR_ACCESS: Imposible obtener privilegios de escritura en el fichero de log.

- Ejemplo:

```
v3_LOG log;
v3MemSet( &log, v3u64u32( sizeof( v3_LOG ) ), 0 );
v3LogOpen( &log, (ascii *) "file.log", 5, V3_LOG_MODE_NUMBER );
/* check for errors */
v3Log( &log, 3, V3_LOG_NOECHO, (ascii *)This is logged. 3 is less than 5.\n" );
v3Log( &log, 8, V3_LOG_NOECHO, (ascii *)"This is not logged. 8 is greater than 5.\n" );
v3LogClose( &log );
v3LogOpen( &log, (ascii *) "file.log", 0x55, V3_LOG_MODE_BITS );
/* check for errors */
v3Log( &log, 0x03, V3_LOG_NOECHO, (ascii *)"This is logged. ( 0x03 & 0x55 ) != 0.\n" );
v3Log( &log, 0x08, V3_LOG_NOECHO, (ascii *)"This is not logged. ( 0x08 & 0x55 ) == 0.\n" );
v3Log( &log, 0x00, V3_LOG_NOECHO, (ascii *)"This is logged. Level 0 is ALWAYS logged.\n" );
v3LogClose( &log );
```

❋ **v3LogClose:**

- Sintaxis:

```
#include "log.h"
s32 v3LogClose( v3_LOG * log );
```

- Descripción:

Cierra el fichero de log que le pasamos como parámetro.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito.

- Errores:

Ninguno.

- Ejemplo:

```
v3_LOG log;
v3MemSet( &log, v3u64u32( sizeof( v3_LOG ) ), 0 );
/* ... */
v3LogClose( &log );
```

3.6.4.11 Funciones para la memoria:

❋ **v3MemAlloc:**

- Sintaxis:

```
#include "cosmmem.h"
void * v3MemAlloc( u64 bytes, u32 secure );
```

- Descripción:

Asigna bloques de memoria como los siguientes: 16*floor((*bytes*+15)/16)) bytes, siendo *bytes* la variable que le pasamos a la función. Si la variable *secure* vale V3_MEM_NORMAL no necesitamos ningún requerimiento especial, mientras que si esta variable vale V3_MEM_SECURE, el sistemas esperará los siguientes aspectos:

- La memoria tiene que ser protegida de cualquier proceso que intente hacer lecturas o escrituras de cualquier proceso que pertenezca al usuario a menos que el proceso posea los privilegios necesarios.
- La memoria no puede ser cambiada o copiada fuera de la memoria activa RAM.
- Las partes del Kernel encargadas de supervisar el espacio no deben ser modificadas.

Si el "flag" de seguridad (*secure*) ha sido forzado, deberemos hacer una llamada a la función v3MemWarning para que esta función nos devuelva un NULL. A partir de este momento, el usuario será avisado en el caso de que el sistema se convierta en inseguro y los programas atenten contra la seguridad de las operaciones.

- Valores devueltos:

La función nos devolverá un puntero apuntando al bloque de memoria RAM, o NULL en caso de error.

- Errores:

Ninguno.

- Ejemplo:

```
u64 size;
void * mem;
_V3_SET64( size, 00000000, 00000400 );
mem = v3MemAlloc( size, V3_MEM_NORMAL );
if ( mem == NULL )
 {
   /* Error */
   return( V3_FAIL );
 }
```

❋ **v3MemRealloc:**

- Sintaxis:

```
#include "cosmmem.h"
void * v3MemRealloc( void * memory, u64 bytes, u32 secure );
```

- Descripción:

Ajusta el tamaño de los bloques de memoria (*memory*) RAM a un tamaño de bytes especificado en la variable *byte*, haciendo copia de cualquier dato existente en el bloque de memoria. Si la variable *memory* vale NULL, entonces esta función tendrá la misma funcionalidad que la función v3MemAlloc. Si la variable *secure* vale V3_MEM_NORMAL, no necesitamos ningún requerimiento especial, mientras que si esta variable vale V3_MEM_SECURE, el sistemas esperará los siguientes aspectos:

- La memoria tiene que ser protegida de cualquier proceso que intente hacer lecturas o escrituras de cualquier proceso que pertenezca al usuario a menos que el proceso posea los privilegios necesarios.
- La memoria no puede ser cambiada o copiada fuera de la memoria activa RAM.
- Las partes del Kernel encargadas de supervisar el espacio no deben ser modificadas.

Si el "flag" de seguridad (*secure*) ha sido forzado, deberemos hacer una llamada a la función v3MemWarning para que esta función nos devuelva un NULL. A partir de este momento, el usuario será avisado en el caso de que el sistema se convierta en inseguro y los programas atenten contra la seguridad de las operaciones. Si la variable *bytes* vale 0, entonces se liberará cualquier espacio de memoria y la función nos devolverá un NULL. Cualquier espacio de memoria expandida contendrá valores desconocidos.

- Valores devueltos:

La función nos devolverá un puntero apuntando al nuevo espacio de memoria o NULL si es imposible cambiar el tamaño de la memoria, dejando el antiguo espacio de memoria intacto.

- Errores:

Ninguno.

- Ejemplo:

```
u64 size;
void * mem;
void * small;
_V3_SET64( size, 00000000, 00000400 );
mem = v3MemAlloc( size, V3_MEM_NORMAL );
if ( mem == NULL )
  {
   /* Error */
   return( V3_FAIL );
  }
_V3_SET64( size, 00000000, 00000200 );
small = v3MemRealloc( mem, size, 0 );
if ( small == NULL )
  {
   /* Error */
   return( V3_FAIL );
  }
```

❋ **v3MemCopy:**

- Sintaxis:

```
#include "cosmmem.h"
s32 v3MemCopy( void * dest, const void * src, u64 length );
```

- Descripción:

Copia una cantidad determinada de bytes indicada en la variable *length* de la memoria origen (*src*) a un destino (*dest*).

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

Ninguno.

- Ejemplo:

```
void * old;
void * new;
u64 size;
_V3_SET64( size, 00000000, 00000400 );
old = v3MemAlloc( size, V3_MEM_NORMAL );
if ( old == NULL )
  {
   /* Error */
   return( V3_FAIL );
  }
new = v3MemAlloc( size, V3_MEM_NORMAL );
if ( new == NULL )
  {
   /* Error */
   return( V3_FAIL );
  }
if ( v3MemCopy( new, mem, size ) == V3_FAIL )
  {
   /* Error */
   return( V3_FAIL );
  }
```

❋ **v3MemSet:**

- Sintaxis:

```
#include "cosmmem.h"
s32 v3MemSet( void * memory, u64 length, u8 value );
```

- Descripción:

Establece un tamaño (*length*) de bytes de memoria (*memory*) en la variable *value*.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

Ninguno.

- Ejemplo:

```
u64 size;
void * mem;
void * small;
_V3_SET64( size, 00000000, 00000400 );
mem = v3MemAlloc( size, V3_MEM_NORMAL );
if ( mem == NULL )
  {
   /* Error */
   return( V3_FAIL );
  }
if ( v3MemSet( mem, size, 0xFF ) == V3_FAIL )
  {
   /* Error */
   return( V3_FAIL );
  }
```

❋ **v3MemCmp:**

- Sintaxis:

```
#include "cosmmem.h"
s32 v3MemCmp( const void * blockA, const void * blockB, u64 max_bytes );
```

- Descripción:

Compara dos bloques de memoria, *blockA* y *blockB*, sobre un número máximo de bytes indicado en la variable *max_bytes*.

- Valores devueltos:

La función nos devolverá 0 si los dos bloques de memoria son idénticos. Si no son idénticos, nos devolverá un número positivo o negativo dependiendo del resultado de la siguiente operación: *blockA*[x] - *blockB*[x]. Si cualquier parámetro vale NULL ó 0, entonces la función nos devolverá un –1 a menos que los dos bloques a comparar, *blockA* y *blockB*, sean idénticos.

- Errores:

Ninguno.

- Ejemplo:

```
u64 size;
void memA[256];
void memB[256];
s32 diff;
_V3_SET64( size, 00000000, 00000100 );
if ( ( diff = v3MemCmp( memA, memB, size ) ) == 0 )
  {
    /* memory blocks are the same */
  }
```

❋ **v3MemOffset:**

- Sintaxis:

```
#include "cosmmem.h"
void * v3MemOffset( const void * memory, u64 offset );
```

- Descripción:

Calcula la dirección de memoria (*memory*) más el desplazamiento (*offset*).

- Valores devueltos:

La función nos devolverá un puntero a la localización de los datos o NULL si el espacio de direccionamiento se ha excedido.

- Errores:

Ninguno.

- Ejemplo:

```
u64 offset;
void * memory;
void * ptr;
_V3_SET64( offset, 00000000, 00000100 );
ptr = v3MemOffset( memory, offset );
/* ptr is a pointer to 256 bytes past memory */
```

❋ **v3MemFree:**

- Sintaxis:

```
#include "cosmmem.h"
void v3MemFree( void * memory );
```

- Descripción:

Libera el bloque de memoria requerido e indicado en la variable *memory*. Esta variable puede ser NULL.

- Valores devueltos:

Ninguno.

- Errores:

Ninguno.

- Ejemplo:

```
u64 size;
void * mem;
_V3_SET64( size, 00000000, 00000400 );
mem = v3MemAlloc( size, V3_MEM_NORMAL );
if ( mem == NULL )
  {
   /* Error */
   return( V3_FAIL );
  }
v3MemFree( mem );
```

❋ **v3MemSystem:**

- Sintaxis:

```
#include "cosmmem.h"
s32 v3MemSystem( u64 * amount );
```

- Descripción:

Establece un número (*amount*) de bytes de memoria RAM en el sistema.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  La posible causa de fallo es la siguiente:
  - El sistema operativo no soporta esta función.

- Ejemplo:

```
u64 memory_size;
void * mem;
if ( v3MemSystem( &memory_size ) != V3_PASS )
  {
    /* couldn't get memory total */
  }
```

❋ **v3MemWarning:**

- Sintaxis:

```
#include "cosmmem.h"
void * v3MemWarning( void );
```

- Descripción:

  Crea una serie de mensajes informativos para el usuario indicándole que la memoria es segura e informándole de los posibles acontecimientos que se pudieran dar.

- Valores devueltos:

  Ninguno.

- Errores:

  Ninguno.

- Ejemplo:

```
v3MemWarning();
```

❋ **v3MemDumpLeaks:**

- Sintaxis:

```
#include "cosmmem.h"
s32 v3MemDumpLeaks( const ascii * filename );
```

- Descripción:

Escribe una lista de las partes de memoria que aún no han sido liberadas por la función v3MemFree en un fichero que vienen indicado en la variable *filename*.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

Cualquier error relativo a la apertura de un fichero.

- Ejemplo:

```
v3MemDumpLeaks( (ascii *) "leaks.txt" );
```

3.6.4.12 Funciones de red:

❋ **v3NetOpen:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetOpen( v3_NET * net, u32 host, u16 port, u32 my_host,
u16 my_port, u32 mode, u32 firehost, u16 fireport, const ascii * firepass );
```

- Descripción:

Abre una conexión de red en un equipo remoto (host) a través de un puerto (*port*). Si la variable *mode* vale V3_NET_MODE_TCP, eso nos está indicando que el protocolo del que vamos a hacer uso es TCP; en cambio, si la variable *mode* vale V3_NET_MODE_UDP, nos está indicando que el protocolo será UDP. Las variables *firehost, fireport, firepass* nos sirven para los proxies que necesitan atravesar los firewalls.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_NET_ERROR_HOST: Imposible conectar con el equipo.

  V3_NET_ERROR_PORT: Imposible escuchar o conectar con el puerto.

  V3_NET_ERROR_FIREHOST: Equipo firewall incorrecto.

  V3_NET_ERROR_FIREPORT: Puerto del firewall incorrecto.

  V3_NET_ERROR_FIREPASS: Contraseña de firewall incorrecta.

  V3_NET_ERROR_CLOSED: Conexión cerrada.

  V3_NET_ERROR_NO_NET: Sin soporte de red.

- Ejemplo:

```
u32 ip;
s32 result;
v3_NET * net, * net2;
/* ... */
result = v3NetOpen( net, ip, 80, 0, 0, V3_NET_MODE_TCP, 0, 0, NULL);
if ( result == V3_PASS )
  v3PrintA( "Connection to port 80 established.\n" );
else
   v3PrintA( "Connection to port 80 failed.\n" );
/* same thing with macro */
result = _V3_NETOPEN( net2, ip, 80 );
if ( result == V3_PASS )
   v3PrintA( "Another connection to port 80 established.\n" );
else
   v3PrintA( "Another connection to port 80 failed.\n" );
```

❋ **v3NetSend:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetSend(v3_NET * net,u32* bytes_sent,const void * data,u32 length );
```

- Descripción:

Envía los datos especificados en la variable *data* a través de la conexión de red que se mantiene abierta. La variable *bytes_sent* nos indica el número de bytes que hemos enviado hasta el momento. La conexión debe abrirse bajo el modo V3_NET_MODE_TCP.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_NET_ERROR_HOST: Imposible conectar con el equipo.

V3_NET_ERROR_CLOSED: Conexión cerrada.

V3_NET_ERROR_NO_NET: Sin soporte de red.

- Ejemplo:

```
s32 result;
v3_NET * net;
ascii to_send[30];
u32 bytes;
/* ... */
v3StrCopy( to_send, "Hello world!\n", v3u64u32( 30 ) );
result=v3NetSend(net,&bytes,to_send,v3u32u64(v3StrLengthA( to_send ) ) );
if ( result == V3_PASS )
  {
   v3PrintA( "String sent to remote host\n" );
  }
else
  {
   v3PrintA( "Sent %u bytes of string\n", bytes );
  }
```

❋ **v3NetRecv:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetRecv( void * buffer, u32 * bytes_received, v3_NET * net,
u32 length, u32 wait_ms );
```

- Descripción:

Lee cualquier dato que se encuentre disponible cuyo tamaño sea el indicado en la variable *length* del buffer (*buffer*). Si la variable *wait_ms* es distinta de 0, entonces el retardo en milisegundos se establecerá a partir de lo que se tarde en recibir todos los datos estipulados en los parámetros de la función. La variable *bytes_recived* nos indica el número de bytes que se han leído hasta el momento. La conexión debe abrirse bajo el modo V3_NET_MODE_TCP. Un buen tiempo de retardo para introducir en la variable *wait_ms* podría ser *length*+5000, es decir, daremos 5 segundos más 1 segundo/KB de datos, siendo esto un tiempo suficiente para transferir los datos entre los modems y diferentes redes existentes.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_NET_ERROR_HOST: Imposible conectar con el equipo.

V3_NET_ERROR_CLOSED: Conexión cerrada.

V3_NET_ERROR_NO_NET: Sin soporte de red.

- Ejemplo:

```
u32 bytes;
v3_NET * net;
ascii getbuff[1024];
/* ... */
while ( v3NetRecv( getbuff, &bytes, net, 1024,1024 + 5000 ) == V3_PASS )
  {
    getbuff[bytes] = 0;
    v3PrintA( "%s", getbuff );
  }
v3printA( "\nThe remote host closed the connection.\n" );
```

❋ **v3NetSendUDP:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetSendUDP( v3_NET * net, u32 host, u16 port,
const void * data, u32 length );
```

- Descripción:

Se envían los datos establecidos en la variable *data* al equipo (*host*) y a un puerto determinado (*port*). Al utilizar un protocolo no orientado a conexión como es UDP, no se asegura que los datos vayan a llegar. La conexión debe abrirse bajo el modo V3_NET_MODE_UDP.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_NET_ERROR_NO_NET: Sin soporte de red.

❋ **v3NetRecvUDP:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetRecvUDP( void * buffer, u32 * bytes_read, v3_NET * net, u32
length,  v3_NET_ACL * acl, u32 wait_ms );
```

- Descripción:

Lee paquetes UDP dentro del buffer (*buffer*). Si la longitud del buffer no es suficiente para la longitud de los paquetes, el resto de los datos serán descartados. Si la variable *wait_ms* es distinta de 0, nos indicará el retardo que tendremos entre paquete y paquete. La conexión debe abrirse bajo el modo V3_NET_MODE_UDP.

- Valores devueltos:

La función nos devolverá la variable *length* en caso de éxito o nos devolverá 0 o un valor menor de *length* en caso de error. La variable *last_error* también se enviará como código de error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error para la variable *last_error*:

  V3_NET_ERROR_NO_NET: Sin soporte de red.

❋ **v3NetListen:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetListen( v3_NET * net, u32 ip, u16 port, u32 mode, u32 queue );
```

- Descripción:

  Establecerá un punto de escucha en el puerto indicado dentro de la variable *port* para que acepte conexiones. Si es posible, escuchará en una determinada IP indicada en la variable *ip*. Si esta variable vale 0, escuchará en todos los interfaces. Si la variable *mode* vale V3_NET_MODE_TCP, eso nos está indicando que el protocolo del que vamos a hacer uso es TCP; en cambio, si la variable *mode* vale V3_NET_MODE_UDP, nos está indicando que el protocolo será UDP. Si la variable *port* vale 0, será el sistema operativo el que nos asigne un puerto. En la variable *queue* tenemos representada una cola para poder encolar las diferentes peticiones de conexión que nos vayan llegando.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

  Se pueden dar las siguientes salidas como códigos de error:

  V3_NET_ERROR_PORT: Imposible escuchar o conectar con el puerto.

  V3_NET_ERROR_NO_NET: Sin soporte de red.

- Ejemplo:

```
s32 result;
v3_NET * net;
result = v3NetListen( net, 0, 5150, V3_NET_MODE_TCP, 5 );
if ( result == V3_PASS )
  v3PrintA( (ascii *)"Listening on port 5150 on all interfaces.\n" );
else
  v3PrintA( (ascii *)"Couldn't bind to port 5150. Is the port in use?\n" );
```

❋ **v3NetAccept:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetAccept( v3_NET * new_connection, v3_NET * net, v3_NET_ACL
* acl, u32 wait );
```

- Descripción:

Aceptará una conexión de red, si es que se estuviera esperando alguna. Si la variable *wait*, vale V3_NET_ACCEPT_WAIT eso nos indica que no se devolverá nada hasta que no se acepte una conexión; en cambio, si vale V3_NET_ACCEPT_NOWAIT, se devolverá inmediatamente.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_NET_ERROR_CLOSED: Conexión cerrada.

V3_NET_ERROR_NO_NET: Sin soporte de red.

- Ejemplo:

```
s32 result;
v3_NET * net;
v3_NET * connection;
v3NetListen( net, 0, 5150, 5 );
/* ... */
result=v3NetAccept(connection,net, NULL, V3_NET_ACCEPT_NOWAIT );
if ( result == V3_PASS )
   v3PrintA( "Connection received to port 5150.\n" );
else
   v3PrintA( "Nobody is waiting to connect on port 5150.\n" );
```

❋ **v3NetClose:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetClose( v3_NET * net );
```

- Descripción:

Cierra una conexión y limpia todos los datos que puedan quedar de la misma.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un código de error si se ha producido un error.

- Errores:

Se pueden dar las siguientes salidas como códigos de error:

V3_NET_ERROR_CLOSED: Conexión cerrada.

V3_NET_ERROR_NO_NET: Sin soporte de red.

- Ejemplo:

```
s32 result;
v3_NET * net;
/* ... */
result = v3NetClose( net );
if ( result == V3_PASS )
   v3PrintA( "Connection closed and remaining data flushed.\n" );
else
   v3PrintA( "The remote host closed the connection before us.\n" );
```

❋ **v3NetDNS:**

- Sintaxis:

```
#include "cosmnet.h"
u32 v3NetDNS( u32 * ip, u32 length, ascii * name );
```

- Descripción:

Hace las consultas DNS necesarias. Establece la longitud (*length*) de los nombres (*name*) de los equipos para las IP dadas a la función (*ip*).

- Valores devueltos:

La función nos devolverá el número de IP´s establecidas ó 0 en caso de fallo.

- Errores:

  Las posibles causas de fallo son las siguientes:
  - *ip* o *name* son NULL.
  - *length* es 0.
  - No se puede obtener la IP a través del nombre del equipo.

- Ejemplo:

```
u32 hosts[16];
u32 result;
result = v3NetDNS( &host, 16, "cosm.mithral.com" );
if ( result > 0 )
   v3PrintA( "Hostname cosm.mithral.com successfully resolved.\n" );
else
  v3PrintA( "Error resolving host cosm.mithral.com.\n" );
```

❋ **v3NetRevDNS:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetRevDNS( v3_NET_HOSTNAME * name, u32 ip );
```

- Descripción:

  Hace el DNS inverso. Establece los nombres a partir de las IP´s de los equipos.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Las posibles causas de fallo son las siguientes:
  - *ip* o *name* son NULL.
  - *length* es 0.
  - No se puede obtener el dominio del nombre para la IP (*ip*).

- Ejemplo:

```
s32 result;
u32 ip = 0xD162B486;
v3_NET_HOSTNAME hostname;
result = v3NetRevDNS( hostname, ip );
if ( result == V3_PASS )
   v3PrintA( (ascii *) hostname );
else
   v3PrintA( "Unable to perform reverse DNS.\n" );
```

❋ **v3NetMyIP:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetMyIP( u32 * ip );
```

- Descripción:

Establece la dirección IP para el equipo IP dado en la variable *ip*.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

Las posibles causas de fallo son las siguientes:

- *ip* es NULL.
- No se puede conseguir el nombre del equipo local.
- No se puede obtener la IP para el equipo local.

- Ejemplo:

```
s32 result;
u32 host;
result = v3NetMyIP( &host );
if ( result == V3_PASS )
   v3PrintA( "Local IP address successfully determined.\n" );
else
   v3PrintA( "Unable to determine local IP address.\n" );
```

❋ **v3NetACLAdd:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetACLAdd( v3_NET_ACL * acl, u32 ip, u32 mask, u32 permission,
v3_time expires );
```

- Descripción:

Añade la IP y la máscara de red (*ip/Mask*) al ACL (*acl*). La variable *permission* debería valer V3_NET_ALLOW ó V3_NET_DENY. La variable *expires* nos indica el tiempo en el que, a partir de él, cualquier dato que entre será borrado automáticamente. Este tiempo debería estar basado en la función v3SystemClock.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

Ninguno.

❋ **v3NetACLDelete:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetACLDelete( v3_NET_ACL * acl, u32 ip, u32 mask );
```

- Descripción:

Borra la IP (ip) y la máscara (*mask*) de ACL (*acl*).

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

Ninguno.

❋ **v3NetACLTest:**

- Sintaxis:

```
#include "cosmnet.h"
s32 v3NetACLTest( v3_NET_ACL * acl, u32 ip );
```

- Descripción:

Comprueba si alguna dirección IP es aceptada por el ACL (*acl*). Se acepta una dirección IP si pasa todas y cada una de las entradas del ACL en orden y tiene además un modo V3_NET_ALLOW.

- Valores devueltos:

La función nos devolverá V3_NET_ALLOW si se permite el paso o V3_NET_DENY si no se le permite el paso.

- Errores:

Ninguno.

❋ **v3NetACLFree:**

- Sintaxis:

```
#include "cosmnet.h"
void v3NetACLFree( v3_NET_ACL * acl );
```

- Descripción:

Libera los datos internos del ACL.

- Valores devueltos:

Ninguno.

- Errores:

Ninguno.

3.6.4.13 Funciones para sistema "Benchmark":

- **v3SpeedFloat:**
  - Sintaxis:

    ```
    #include "speed.h"
    u64 v3SpeedFloat( void );
    ```

  - Descripción:

    Establece un número de coma flotante usando 64 bits matemáticos. Esta función trabaja con multiplicaciones de matrices expandiendo los datos con 64KB de datos aproximadamente. A esta función le puede llevar un poco de tiempo procesar el resultado y consumir bastante CPU.

  - Valores devueltos:

    La función nos devolverá el número de coma flotante.

  - Errores:

    Ninguno.

  - Ejemplo:

    ```
    u64 fspeed;
    fspeed = v3SpeedFloat();
    ```

- **v3SpeedInt:**
  - Sintaxis:

    ```
    #include "speed.h"
    u64 v3SpeedInt( void );
    ```

  - Descripción:

    Realiza las operaciones usando 32 bits de enteros. Las operaciones las realiza multiplicando matrices expandiendo los datos de 48KB aproximadamente. A esta función le puede llevar un poco de tiempo procesar el resultado y consumir bastante CPU.

  - Valores devueltos:

    La función nos devolverá las transformaciones a enteros.

  - Errores:

    Ninguno.

- Ejemplo:

```
u64 ispeed;
ispeed = v3SpeedInt();
```

3.6.4.14 Funciones para el tiempo:

❋ **v3Time:**

- Sintaxis:

```
#include "cosmtime.h"
s32 v3Time( v3_time * dest, const v3_TIME_CORRECTION * const
corrections );
```

- Descripción:

Obtiene la hora actual basándose en la hora que posea la red ya sincronizada para tal efecto.

- Valores devueltos:

La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

Las posibles causas de fallo son las siguientes:

- *dest* o *corrections* son NULL.
- La variable *corrections* no ha sido correctamente inicializada.
- No se pude obtener la hora del sistema.

❋ **v3TimeSet:**

- Sintaxis:

```
#include "cosmtime.h"
s32 v3TimeSet( v3_TIME_CORRECTION * corrections,u32 * ip_list, u32
ip_count );
```

- Descripción:

Se sincroniza el tiempo con el tiempo de la red. Tiene una lista de servidores de hora (*ip_list*) para sincronizarse con ellos.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Ninguno.

❋ **v3TimeUnitsGregorian:**

- Sintaxis:

  ```
  #include "cosmtime.h"
  s32 v3TimeUnitsGregorian( v3_TIME_UNITS * units, v3_time time );
  ```

- Descripción:

  Rellena la estructura *units* con la hora (*time*) en formato Gregoriano. La estructura de v3_TIME_UNITS tiene el siguiente formato:

  ```
  typedef struct
  {
  /* all values are zero based */
  s64 year;   /* year */
  u32 month;  /* month */
  u32 day;    /* day of month */
  u32 wday;   /* day of week */
  u32 yday;   /* day of year */
  u32 hour;   /* hours */
  u32 min;    /* minutes */
  u32 sec;    /* seconds */
  u64 subsec; /* sub-seconds, note: unsigned unlike v3_time */
  } v3_TIME_UNITS;
  ```

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  La posible causa de fallo es la siguiente:
  - *units* es NULL.

- Ejemplo:

```
v3_time mytime;
v3_TIME_UNITS myunits;
const ascii * days[7] = V3_TIME_DAYS3;
const ascii * months[12] = V3_TIME_MONTHS;
u64 milli;
_V3_SET64( milli, 00418937, 4BC6A7EF );
if ( v3SystemClock( &mytime ) == V3_PASS )
{
   if ( v3TimeUnitsGregorian( &myunits, mytime ) == V3_PASS )
            {
              v3PrintA( (ascii *)"%.9s %.9s %u, %j %02u:%02u:%02u.%03v
              UTC, d%u\n",days[myunits.wday], months[myunits.month],
              myunits.day + 1,myunits.year, myunits.hour, myunits.min,
              myunits.sec,v3u64Div( myunits.subsec, milli ), myunits.yday +
              1);;
            }
}
```

❋ **v3TimeDigestGregorian:**

- Sintaxis:

```
#include "cosmtime.h"
s32 v3TimeDigestGregorian( v3_time * time,const v3_TIME_UNITS * const
units );
```

- Descripción:

  Convierte al formato pedido por la función v3_Time valores en formato Gregoriano.

- Valores devueltos:

  La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá V3_FAIL si se ha producido un error.

- Errores:

  Las posibles causas de fallo son las siguientes:
  - *Time* o *units* son NULL.
  - Mes o número de días incorrectos en la variable *units*.

3.6.4.15 Funciones del programa principal y de "autotest":

- **v3Test:**
  - Sintaxis:

```
#include "cosm.h"
s32 v3Test( s32 * failed_module, s32 * failed_test, s32 module_num );
```

  - Descripción:

    Si la variable *module_num* vale 0, entonces comprueba todas las funciones de la plataforma y devolverá un valor negativo correspondiente al módulo que falla. Las variables *failed_module* y *failed_test* serán las encargadas de indicar cuales son las funciones que están fallando.

  - Valores devueltos:

    La función nos devolverá V3_PASS si se ha llevado a cabo con éxito, o nos devolverá un número negativo correspondiente al módulo que está fallando si se ha producido un error.

  - Errores:

    Un número negativo será devuelto en caso de que algún módulo no funcione correctamente.

  - Ejemplo:

```
const ascii * test_modules[] = V3_TEST_MODULES;
s32 error;
s32 error2;
v3PrintA( (ascii *) "Running system tests... " );
if ( v3Test( &error, &error2, 0 ) != V3_PASS )
  {
    v3PrintA( (ascii *) "Self-test failure in module %.16s %i.\n",
    test_modules[( -error > V3_TEST_MODULE_MAX ) ? 0 : -error],error2 );
    v3ProcessEnd( error );
  }
v3PrintA( (ascii *) "all passed.\n" );
```

3.6.4.16 Definiciones y tipos:

Las definiciones más importantes que hay que reseñar son las siguientes:

CPU_TYPE:

Los tipos de CPU que pueden ser utilizados en este sistema están gestionados bajo este tipo de datos. Los valores que puede adoptar éste van desde 0 hasta V3_CPU_TYPE_MAX, ambos inclusive. Los valores que contiene el array V3_CPU_TYPES están llenos de nombres que se pueden usar. Los valores de los que estamos hablando son los siguientes:

- CPU_UNKNOWN
- CPU_X86
- CPU_POWERPC
- CPU_MIPS
- CPU_MIPS64
- CPU_ALPHA
- CPU_PA_RISC
- CPU_68K
- CPU_SPARC
- CPU_SPARC64
- CPU_POWER
- CPU_VAX
- CPU_ARM
- CPU_88K
- CPU_S390
- CPU_SH4

OS_TYPE:

Los tipos de Sistema Operativo bajo los que correrá este sistema están gestionados bajo este tipo de datos. Los valores que puede adoptar éste van desde 0 hasta V3_OS_TYPE_MAX, ambos inclusive. Los valores que contiene el array V3_OS_TYPES están llenos de nombres que se pueden usar. Los valores de los que estamos hablando son los siguientes:

- OS_UNKNOWN
- OS_WIN32
- OS_MACOS
- OS_MACOSX
- OS_LINUX
- OS_BSDI
- OS_NETBSD
- OS_FREEBSD
- OS_OPENBSD
- OS_NEXTSTEP
- OS_BEOS

- OS_IRIX
- OS_IRIX64
- OS_SUNOS
- OS_SOLARIS
- OS_SCO
- OS_QNX
- OS_TRU64
- OS_VMS
- OS_OS2
- OS_UNIXWARE
- OS_HPUX
- OS_MACH
- OS_AIX
- OS_AUX
- OS_AMIGAOS
- OS_NETWARE
- OS_MVS
- OS_ULTRIX
- OS_DGUX
- OS_SINIX
- OS_DYNIX
- OS_OS390
- OS_RISCOS
- OS_OS9

A la hora de compilar la plataforma Cliente/Servidor le podemos otorgar o quitar unas funcionalidades a la misma. Esto lo hacemos a través de unos parámetros funcionales que se utilizan a la hora de compilar el software. Estos parámetros se encuentran explicados en las siguientes definiciones opcionales:

WITH_GUI: Compilaremos la plataforma Cliente/Servidor con soporte para el Interfaz Gráfico de Usuarios.

NO_THREADS: La plataforma Cliente/Servidor será compilada sin soporte para threads.

NO_IEEE_FLOAT: La plataforma Cliente/Servidor será compilada sin soporte para la notación en coma flotante propuesta por el IEEE.

NO_NETWORKING: La plataforma Cliente/Servidor será compilada sin soporte para las funciones de red.

NO_CRYPTO: La plataforma Cliente/Servidor será compilada sin soporte para las funciones criptográficas y de resumen (hash).

NONPORTABLE_CODE: La librería cosm.h redefinirá funciones peligrosas o reemplazadas del código estándar de C. Para usar este tipo de funciones será necesario compilar la plataforma con este parámetro, pero hay que tener en cuenta que el código no será portable a diferentes plataformas y el usuario se quedará sin la posibilidad de ser

avisado, por parte de la aplicación, de que está usando funciones de este tipo y que pueden ser peligrosas para la portabilidad de la aplicación.

MEM_LEAK_FIND: Este parámetro habilita la función v3MemDumpLeaks y todas las funciones relacionadas con pérdidas de memoria persistentes.

NET_LOG_PACKETS: Este parámetro habilita paquetes para ficheros de log.

En esta plataforma se encuentran definidos una serie de datos que son portables para la mayoría de plataformas y Sistemas Operativos existentes en la actualidad. Los tipos de datos a los que estamos haciendo mención son los siguientes:

Tipos de números enteros:

u8:	Integer sin signo de 8-bit.
s8:	Integer con signo de 8-bit.
u16:	Integer sin signo de 16-bit.
s16:	Integer con signo de 16-bit.
u32:	Integer sin signo de 32-bit.
s32:	Integer con signo de 32-bit.
u64:	Integer sin signo de 64-bit.
s64:	Integer con signo de 64-bit.
u128:	Integer sin signo de 128-bit.
s128:	Integer con signo de 128-bit.

Tipos de números con notación de coma Flotante:

f32:	Coma flotante de 32-bit.
f64:	Integer con signo de 64-bit.

Tipo Texto:

ascii:	Maneja un carácter ASCII.
unicode:	Maneja un carácter UNICODE.

Tipo Tiempo:

v3_Time: El formato de este tipo de datos es el siguiente: 0 = 00:00:00 UTC, Jan 1, 2000 AD. Esto nos concede un rango de años igual a +/- 2.923E11 y una resolución de 5.421E-20 segundos.

En la librería cputypes.h, como hemos comentado anteriormente, tenemos definidos dos arrays de caracteres para la gestión de CPU´s y de Sistemas Operativos como son V3_CPU_TYPES y V3_OS_TYPES. La mayoría de ellos son para salidas del "debugger", no siendo relevantes para la mayor parte de los casos. Un ejemplo ilustrativo para entender mejor este tipo de datos sería el que se presenta a continuación:

```
const ascii * cpu_types[] = V3_CPU_TYPES;
const ascii * os_types[] = V3_OS_TYPES;
v3PrintA( (ascii *) "CPU = %.20s, OS = %.20s\n",
cpu_types[( CPU_TYPE > V3_CPU_TYPE_MAX ) ? 0 : CPU_TYPE],
os_types[( OS_TYPE > V3_OS_TYPE_MAX ) ? 0 : OS_TYPE] );
```

Para que el sistema nos pueda indicar cuántas máquinas tenemos corriendo a la misma vez dentro de la plataforma Cliente/Servidor, tenemos dedicado el Tipo ENDIAN con las siguientes posibilidades:

- V3_ENDIAN_CURRENT: Este tipo nos indica cuántas máquinas tenemos corriendo a la vez dentro del sistema.
- V3_ENDIAN_BIG
- V3_ENDIAN_LITTLE

Un ejemplo ilustrativo para este tipo de datos sería el siguiente:

```
if ( V3_ENDIAN_CURRENT == V3_ENDIAN_BIG )
  {
    /* do the big thing */
  }
else
  {
    /* do the little thing */
   }
```

Para la mayoría de los valores que devuelven las funciones tenemos definidos tres tipos de datos que son:

- V3_PASS: Nos indica que la operación se ha realizado con éxito.
- V3_FAIL: Nos indica que la operación no se ha realizado con éxito.
- NULL: Nos indica que el valor contenido por la variable o función es nulo.

_V3_SET

- Descripción:

  Establece o inicializa un entero de 64 ó 128 bits a un valor constante. El primer argumento para esta función sería la variable a establecer, seguida del valor en hexadecimal sin el prefijo “0x”, en grupos de 32 bits, dos en total para el caso de 64 bits y cuatro en total para el caso de 128 bits.

- Ejemplo para el caso de 64 bits:

```
u64 a;
s64 b;
_V3_SET64( a, 4FF46423, 12345678 );
/* a is now 0x4FF4642312345678 */
_V3_SET64( b, 49FDC238, 87654321 );
/* b is now 0x49FDC23887654321 */
```

- Ejemplo para el caso de 128 bits:

```
u128 a;
s128 b;
_V3_SET128( a, 01234567, 89ABCDEF, FEDCBA98, 76543210 );
/* a is now 0x0123456789ABCDEFFEDCBA9876543210 */
_V3_SET128( b, 32507DFF, DAF85A34, 7AF51C34, 45A54391 );
/* b is now 0x32507DFFDAF85A347AF51C3445A54391 */
```

_V3_EQ

- Descripción:

  Compara un entero de 64 ó 128 bits en un valor constante. Devolverá un valor distinto de 0 cuando sean iguales y devolverá un 0 en caso de que no lo sean. El primer argumento para esta función sería la variable a comparar, seguida de la variable con la que se tiene que comparar la primera en hexadecimal sin el prefijo “0x”, en grupos de 32 bits, dos en total para el caso de 64 bits y cuatro en total para el caso de 128 bits.

- Ejemplo para el caso de 64 bits:

```
u64 a;
s64 b;
_V3_SET64( a, 4FF46423, 12345678 );
if ( _V3_EQ64( a, 4FF46423, 12345678 ) )
  {
    /* equal */
  }
_V3_SET64( b, 49FDC238, 87654321 );
if ( _V3_EQ64( b, 49FDC238, 87654321 ) )
  {
    /* equal */
  }
```

- Ejemplo para el caso de 128 bits:

```
u128 a;
s128 b;
_V3_SET128( a, 01234567, 89ABCDEF, FEDCBA98, 76543210 );
if ( _V3_EQ128( a, 01234567, 89ABCDEF, FEDCBA98, 76543210 ) )
  {
    /* equal */
  }
_V3_SET128( b, 32507DFF, DAF85A34, 7AF51C34, 45A54391 );
if ( _V3_EQ128( a, 32507DFF, DAF85A34, 7AF51C34, 45A54391 ) )
  {
    /* equal */
  }
```

3.6.5 Funciones Relevantes:

3.6.5.1 Introducción:

En este apartado que nos ocupa, vamos a proceder a presentar aquellas funciones que resultan más importantes o/e interesantes desde el punto de vista de la facilidad de uso y portabilidad de esta plataforma Cliente/Servidor. La visión que se pretende dar es que, con el cambio de unas pocas funciones, una empresa o cualquier otra institución (léase una universidad, un hospital, un laboratorio químico, etc...), en definitiva, todos aquellos que quieran hacer uso de una distribución de trabajo, podrán obtener una plataforma que les permita distribuir la carga de trabajo sin necesidad de mucho esfuerzo ni de una importante inversión económica.

Para ello, nos vamos a centrar en esta serie de funciones, dejando de lado a otras que se ocupan del tema de gestión de hilos, semáforos, comunicaciones, etc... Daremos una visión de las funciones más interesantes a modificar, de modo que con esa modificación se consiga amoldar la plataforma Cliente/Servidor a cada una de las situaciones que se presenten.

3.6.5.2 Código fuente:

En primer lugar veremos las funciones que son relevantes desde el punto de vista del Servidor. La misión del servidor en esta filosofía de distribución de carga de trabajo no es más que trocear el trabajo a solventar y dárselo a cada cliente que en ese momento se encuentre conectado a él.

Las funciones a las que estamos haciendo referencia son las que exponemos a continuación:

- *Función Assign*:

Esta función nos permite trocear la carga de trabajo en el servidor y mandársela a los clientes que realicen una petición al estar conectados. Este envío de trabajo se realiza a través de una estructura de datos, que en este caso se llama PACKET_ASSIGNMENT, estructura que contiene toda la información relevante para el cliente que se va a encargar de procesar el cálculo.

Desde el punto de vista de la modificación de la plataforma para diferentes fines, en esta función tan sólo habría que modificar estas dos líneas que se presentan a continuación:

```
assign.numero_cliente = collatz;
v3PrintA( (ascii *) "Mandamos un %u\n", collatz );
```

Para ello, quizás hubiera que cambiar también la estructura PACKET_ASSIGNMENT para que tuviera toda la información que deseamos transmitir al cliente que se va a encargar de resolver el problema que se le asigne.

PACKET_ASSIGNMENT tiene un aspecto que procedemos a presentar a continuación:

```
typedef struct
{
  u32 type;
  /* project assignment data */
  /* número para el 3x+1 */
  u32 numero_cliente;
  u32 start;
  u32 end;
} PACKET_ASSIGNMENT;
```

En esta estructura tenemos una serie de datos que deberían ser transparentes para la/s persona/s que le/s interese modificar la plataforma y moldearla para sus necesidades, exceptuando la variable *u32 numero_cliente*. Esta variable, en el ejemplo que he presentado, sería el valor que tiene que calcular el cliente cuando reciba el trabajo. Si fuera necesario añadir más variables por la magnitud del trabajo a distribuir, se puede realizar sin problemas.

El resto de campos de la estructura son campos que se utilizan en múltiples funciones del código del programa, pero que deben ser transparentes para facilitar así el uso y modificación de esta plataforma.

El código fuente de la función Assign es el que se presenta a continuación:

```
void Assign( v3_NET * net, v3_HTTPD_REQUEST * http_req, u32 use_http, u32
collatz )
{
 PACKET_REQUEST request;
 PACKET_ASSIGNMENT assign;
 u32 tmp_count;
 u32 length;
 u32 start;
 u32 bytes;

 length = (u32) ( sizeof( PACKET_REQUEST ) - sizeof( request.type ) );
 if ( use_http )
 {
  if ( ( v3HTTPDRecv( &request.pad, &bytes, http_req, length,
    length + 5000 ) != V3_PASS ) || ( bytes != length ) )
  {
    v3PrintA( (ascii *) "Network Recv Failed\n" );
    return;
  }
 }
 else
 {
  if ( ( v3NetRecv( &request.pad, &bytes, net, length,
    length + 5000 ) != V3_PASS ) || ( bytes != length ) )
  {
    v3PrintA( (ascii *) "Network Recv Failed\n" );
    return;
  }
 }

 v3u64Load( &request.speed, (u8 *) &request.speed );

 /* Código para 3x+1 */
 assign.numero_cliente = collatz;
 v3PrintA( (ascii *) "Mandamos un %u\n", collatz );
 assign.type = PACKET_TYPE_ASSIGNMENT;
 v3u32Save( (u8 *) &assign.type, &assign.type );
 /* work with the global variables */
 v3MutexLock( &__mutex_assign, V3_MUTEX_WAIT );
 v3u32Save( (u8 *) &assign.start, &__count );
 start = __count;
 if ( start >= __count_end )
 {
  tmp_count = 0;
 }
 else
 {
  tmp_count = (__count + v3u32u64( request.speed )+1);
```

```
    if ( tmp_count >= __count_end )
    {
      __count = __count_end;
      tmp_count = __count_end - 1;
    }
    else
    {
      __count = tmp_count;
      tmp_count--;
    }
  }
  v3MutexUnlock( &__mutex_assign );

  if ( tmp_count != 0 )
  {
    v3PrintA( (ascii *) "Assigned work %08X-%08X\n", start, tmp_count );
  }

  v3u32Save( (u8 *) &assign.end, &tmp_count );

  if ( use_http )
  {
    if ( v3HTTPDSend( http_req, &assign, (u32) sizeof( PACKET_ASSIGNMENT ) )
      != V3_PASS )
    {
      v3PrintA( (ascii *) "Network Send Failed\n" );
      return;
    }
  }
  else
  {
    if ( v3NetSend( net, &bytes, &assign, (u32) sizeof( PACKET_ASSIGNMENT ) )
      != V3_PASS )
    {
      v3PrintA( (ascii *) "Network Send Failed\n" );
      return;
    }
  }
}
```

- *Función Accept*:

Esta función nos permite aceptar la solución dada por un cliente al cual se le asignó un trabajo en un momento anterior. Este envío de resultados al servidor se realiza a través de una estructura de datos, que en este caso se llama PACKET_RESULTS_DYA, estructura que contiene toda la información relevante para que el servidor sea capaz de procesar ese resultado y seguir mandando más carga de trabajo a los diferentes clientes que se encuentren conectados en ese momento.

PACKET_RESULTS_DYA tiene un aspecto que procedemos a presentar a continuación:

```
typedef struct
{
 /* resultado devuelto al servidor */
 /* Escrito para el 3x+1 */
 u32 resultado_servidor;
 u32 numero_enviado; /* Para saber de donde procede el resultado*/
 u32 start;
 u32 end;
 u32 cpu;
 u32 os;
} PACKET_RESULTS_DYA;
```

En esta estructura los datos más importantes a modificar para los distintos fines serían la variables *u32 resultado_servidor* y *u32 numero_enviado*. El resto son tipos de datos que se usan para el correcto funcionamiento de la plataforma, por lo que nos tendrían que quedar transparentes.

El código fuente de la función Assign es el que se presenta a continuación:

```
void Accept( v3_NET * net, v3_HTTPD_REQUEST * http_req, u32 use_http )
{
 PACKET_RESULTS_DYA results;
 PACKET_ACCEPT work_accept;
 const ascii * os_types[] = V3_OS_TYPES;
 const ascii * cpu_types[] = V3_CPU_TYPES;
 u32 length;
 u32 bytes;
 u64 tmp;

 /* Inicializamos results */
 results.resultado_servidor = 0;
 results.numero_enviado = 0;
 results.start = 0;
 results.end = 0;
 results.cpu = 0;
 results.os = 0;
```

```
/* accept and record results as finished */
  length = (u32) ( sizeof( PACKET_RESULTS_DYA )); /*- sizeof( results.type ) )*/

  if ( use_http )
  {
    if ( ( v3HTTPDRecv( &results.start, &bytes, http_req, length,
      length + 5000) != V3_PASS ) || ( bytes != length ) )
    {
      v3PrintA( (ascii *) "Network Recv Failed\n" );
      return;
    }
  }
  else
  {
    if ( ( v3NetRecv( &results/*&results.start*/, &bytes, net, length,
      length + 5000 ) != V3_PASS ) || ( bytes != length ) )
    {
      v3PrintA( (ascii *) "Network Recv Failed\n" );
      return;
    }
  }

  v3u32Load( &results.resultado_servidor, (u8 *) &results.resultado_servidor);
  v3u32Load( &results.numero_enviado, (u8 *) &results.numero_enviado);
  v3u32Load( &results.start, (u8 *) &results.start );
  v3u32Load( &results.end, (u8 *) &results.end );
  v3u32Load( &results.cpu, (u8 *) &results.cpu );
  v3u32Load( &results.os, (u8 *) &results.os );


  /* ... */
v3PrintA( (ascii *) "El número %u ha llegado a converger a %u \n",
results.numero_enviado, results.resultado_servidor);

  /* work with the global variables */
  v3MutexLock( &__mutex_accept, V3_MUTEX_WAIT );
  v3MutexUnlock( &__mutex_accept );
  work_accept.type = PACKET_TYPE_ACCEPT;
  v3u32Save( (u8 *) &work_accept.type, &work_accept.type );
  work_accept.result = V3_PASS;
  v3u32Save( (u8 *) &work_accept.result, &work_accept.result );

  if ( use_http )
  {
    if ( v3HTTPDSend( http_req, &work_accept, (u32) sizeof( PACKET_ACCEPT ) )
      != V3_PASS )
    {
      v3PrintA( (ascii *) "Network Send Failed\n" );
      return;
    }
```

```
 }
 else
 {
  if ( v3NetSend( net, &bytes, &work_accept, (u32) sizeof( PACKET_ACCEPT ) )
    != V3_PASS )
  {
   v3PrintA( (ascii *) "Network Send Failed\n" );
   return;
  }
 }

 /* log work done */
}
```

Ahora pasaremos a ver las funciones que son relevantes desde el punto de vista del Cliente. La misión del Cliente, en este marco de distribución de carga de trabajo, no es más que resolver el trabajo que le ha sido asignado por parte del servidor y proporcionar a éste la solución encontrada.

Las funciones a las que estamos haciendo referencia son las que exponemos a continuación:

- *Función GetWork*:

Esta función nos permite aceptar el trabajo remitido por el servidor previa petición por parte del cliente. Esta obtención de carga de trabajo del cliente se realiza a través de una estructura de datos, que en este caso se llama PACKET_ACTIVE, estructura que contiene toda la información relevante para que el cliente sea capaz de procesar ese trabajo y poder llegar a una solución con el fin de remitir los resultados al servidor.

PACKET_ACTIVE tiene una estructura de datos, la cual procedemos a presentar a continuación:

```
typedef struct
{
  u32 type;
  /* project temporary data */
  u32 start;
  u32 current;
  u32 end;
  u32 total_cliente;
  u32 state;
  u32 numero_cliente;
} PACKET_ACTIVE;
```

En esta estructura los datos más importantes a modificar para los distintos fines serían la variables *u32 total_cliente* y *u32 numero_cliente*. El resto son tipos de datos que se usan para el correcto funcionamiento de la plataforma, por lo que nos tendrían que quedar transparentes.

El código fuente de la función Assign es el que se presenta a continuación:

```
u32 GetWork( PACKET_ACTIVE * work, u32 count )
{
 u32 i, length, received, bytes;
 v3_NET net;
 v3_HTTP http;
 PACKET_REQUEST request;
 PACKET_ASSIGNMENT assign;
 u32 status;

 received = 0;

 /* open connection to server */
 v3MemSet( &net, v3u64u32( sizeof( v3_NET ) ), 0 );
 v3MemSet( &http, v3u64u32( sizeof( v3_HTTP ) ), 0 );
 if ( Connect( &net, &http ) != V3_PASS )
 {
  return( received );
 }

 /* save network data */
 request.type = PACKET_TYPE_REQUEST;
 v3u32Save( (u8 *) &request.type, &request.type );
 v3u64Save( (u8 *) &request.speed, &__speed );

 /* get up to count assignments */
 for ( i = 0 ; i < count ; i++ )
 {
  if ( work[i].state != STATE_INVALID )
  {
   /* dont overwrite good data, skip to next one */
   continue;
  }

  /* send request */
  if ( __use_http )
  {
   if ( v3HTTPPost( &http, &status, (ascii *) "/", &request,
     (u32) sizeof( PACKET_REQUEST ) ) != V3_PASS )
   {
    v3PrintA( (ascii *) "Network send failure\n" );
    v3HTTPClose( &http );
    return( received );
   }
```

```
  }
  else
  {
   if ( v3NetSend( &net, &bytes, &request,
     (u32) sizeof( PACKET_REQUEST ) ) != V3_PASS )
   {
     v3PrintA( (ascii *) "Network send failure\n" );
     v3NetClose( &net );
     return( received );
   }
  }

  /* read assignment */
  length = (u32) sizeof( PACKET_ASSIGNMENT );
  /* make sure to wait long enough, about 1sec/KB + 5sec */
  if ( __use_http )
  {
   if ( ( v3HTTPRecv( &assign, &bytes, &http, length, length + 5000 )
     != V3_PASS ) || ( bytes != length ) )
   {
     v3PrintA( (ascii *) "Network Recv Timeout\n" );
     v3HTTPClose( &http );
     return( received );
   }
  }
  else
  {
   if ( ( v3NetRecv( &assign, &bytes, &net, length, length + 5000 )
     != V3_PASS ) || ( bytes != length ) )
   {
     v3PrintA( (ascii *) "Network Recv Timeout\n" );
     v3NetClose( &net );
     return( received );
   }
  }
  /* load network data */
  v3u32Load( &assign.type, (u8 *) &assign.type );
  v3u32Load( &assign.start, (u8 *) &assign.start );
  v3u32Load( &assign.end, (u8 *) &assign.end );
  if ( assign.type != PACKET_TYPE_ASSIGNMENT )
  {
   if ( __use_http )
   {
     v3HTTPClose( &http );
   }
   else
   {
     v3NetClose( &net );
   }
   v3PrintA( (ascii *) "Bad block type\n" );
```

```
    return( received );
   }

   if ( assign.end == 0 )
   {
    if ( __use_http )
    {
     v3HTTPClose( &http );
    }
    else
    {
     v3NetClose( &net );
    }
    v3PrintA( (ascii *) "Server out of work\n" );
    return( received );
   }

   /* copy into a PACKET_ACTIVE - the Runable data type */
   work[i].type = PACKET_TYPE_ACTIVE;
   work[i].start = assign.start;
   work[i].current = assign.start;
   work[i].end = assign.end;
   work[i].state = STATE_ALIVE;
   work[i].numero_cliente = assign.numero_cliente;

   v3PrintA( (ascii *) "Receieved assignment block %u/%u\n",
    received + 1, count );
   v3PrintA( (ascii *) "current %08X / end %08X\n",
    work[i].current, work[i].end );

   received++;
  }

  if ( __use_http )
  {
   v3HTTPClose( &http );
  }
  else
  {
   v3NetClose( &net );
  }
  return( received );
}
```

- *Función Run*:

Esta función es la más importante a tener en cuenta a la hora de modificar la plataforma para las distintas necesidades que se puedan presentar. Modificando esta función, podremos obtener una distribución de trabajo para la mayoría de los problemas que se nos presenten de manera fácil y sencilla, puesto que las comunicaciones, hilos, etc..., quedarán totalmente transparentes.

Esta función le permite al cliente procesar el trabajo encargado por el servidor. Toda la información para el correcto cálculo de trabajo viene representada en una estructura de datos que en este caso se llama PACKET_ACTIVE. Es la misma estructura que la comentada en la función anterior (GetWork).

El código fuente de la función Run es el que se presenta a continuación:

```
s32 Run( PACKET_ACTIVE * work, u64 iterations )
{
 u64 loops;
 u32 Resultado;

 if ( work->type != PACKET_TYPE_ACTIVE )
 {
  return( STATE_INVALID );
 }

 if ( work->state != STATE_ALIVE )
 {
  return( STATE_DONE );
 }

 /* while work to do, not interupted, and not over our iteration limit */
 while ( work->current != work->end )
 {
  /*Metemos el conjunto de instrucciones para el 3x+1 */
        Resultado = work->numero_cliente;
 /* Inicializamos el resultado para evitar problemas */
        while (Resultado != 1)
                {
                if (Resultado%2==0) /* Comprobamos que el número sea par */
                        Resultado = Resultado/2;
                else /* El número es impar */
                        Resultado = (3*Resultado)+1;
                }
  /* Para controlar su blucle del work current y el work end */
        work->current = work->end;
        work->total_cliente = Resultado;
}
/* Con esta comparativa puedo hacer el bucle */
 /* if out of work to do */
 if ( work->current == work->end )
```

```
 {
   work->state = STATE_DONE;
  }

  return( STATE_ALIVE );
}
```

- *Función PutWork*:

Esta función nos permite enviar los resultados obtenidos en el Cliente hacia el servidor. Este envío de resultados se realiza a través de una estructura de datos, que en este caso se llama PACKET_RESULTS, estructura que contiene todos resultados que tienen que ser remitidos desde el cliente hacia el servidor.

PACKET_RESULTS tiene una estructura de datos, la cual procedemos a presentar a continuación:

```
typedef struct
{
  u32 type;
  /* project results */
  /* resultado devuelto al servidor */
  /* 3x+1 */
  u32 resultado_servidor;
  u32 numero_enviado; /* Para saber de donde procede el resultado*/
  u32 start;
  u32 end;
  u32 cpu;
  u32 os;
} PACKET_RESULTS;
```

En esta estructura los datos más importantes a modificar para los distintos fines serían la variables *u32 resultado_servidor* y *u32 numero_enviado*. El resto son tipos de datos que se usan para el correcto funcionamiento de la plataforma, por lo que nos tendrían que quedar transparentes.

El código fuente de la función PutWork es el que se presenta a continuación:

```
u32 PutWork( PACKET_ACTIVE * work, u32 count )
{
 u32 i, length, sent, bytes;
 v3_NET net;
 v3_HTTP http;
 PACKET_RESULTS results;
 PACKET_ACCEPT work_accept;
 u32 status;

 sent = 0;

 /* open connection to server */
 v3MemSet( &net, v3u64u32( sizeof( v3_NET ) ), 0 );
 v3MemSet( &http, v3u64u32( sizeof( v3_HTTP ) ), 0 );
 if ( Connect( &net, &http ) != V3_PASS )
 {
  return( sent );
 }

 /* get up to count assignments */
 if ( work[i].state != STATE_DONE )
   {
    /* it's not done, done send this one back, move on to next one */
    i = 0;
   }

   /* save network data */
   results.type = PACKET_TYPE_RESULTS;
   v3u32Save( (u8 *) &results.type, &results.type );
   v3u32Save( (u8 *) &results.resultado_servidor ,&work[i].total_cliente);
   v3u32Save( (u8 *) &results.numero_enviado , &work[i].numero_cliente);
   v3u32Save( (u8 *) &results.start, &work[i].start );
   v3u32Save( (u8 *) &results.end, &work[i].end );
   results.cpu = CPU_TYPE;
   v3u32Save( (u8 *) &results.cpu, &results.cpu );
   results.os = OS_TYPE;
   v3u32Save( (u8 *) &results.os, &results.os );
```

```
/* send request */
if ( __use_http )
{
  if ( v3HTTPPost( &http, &status, (ascii *) "/", &results,
    (u32) sizeof( PACKET_RESULTS ) ) != V3_PASS )
  {
    v3PrintA( (ascii *) "Network send failure\n" );
    v3HTTPClose( &http );
    return( sent );
  }
}
else
{
            if ( v3NetSend( &net, &bytes, &results,
    (u32) sizeof( PACKET_RESULTS ) ) != V3_PASS )
  {
    v3PrintA( (ascii *) "Network send failure\n" );
    v3NetClose( &net );
    return( sent );
  }
}

/* read assignment */
length = (u32) sizeof( PACKET_ACCEPT );
/* make sure to wait long enough, about 1sec/KB + 5sec */
if ( __use_http )
{
  if ( ( v3HTTPRecv( &work_accept, &bytes, &http, length, length + 5000 )
    != V3_PASS ) || ( bytes != length ) )
  {
    v3PrintA( (ascii *) "Network Recv Timeout\n" );
    v3HTTPClose( &http );
    return( sent );
  }
}
else
{
  if ( ( v3NetRecv( &work_accept, &bytes, &net, length, length + 5000 )
    != V3_PASS ) || ( bytes != length ) )
  {
    v3PrintA( (ascii *) "Network Recv Timeout\n" );
    v3NetClose( &net );
    return( sent );
  }
}
```

```
   /* load network data */
   v3u32Load( &work_accept.type, (u8 *) &work_accept.type );
   v3u32Load( &work_accept.result, (u8 *) &work_accept.result );

   if ( work_accept.type != PACKET_TYPE_ACCEPT )
   {
    v3PrintA( (ascii *) "Bad block type\n" );
    if ( __use_http )
    {
     v3HTTPClose( &http );
    }
    else
    {
     v3NetClose( &net );
    }
    return( sent );
   }

   /* if work_accept.result == V3_PASS */
   v3PrintA( (ascii *) "Resultados enviados al servidor \n");
   v3PrintA( (ascii *) "Comienza en  %08X / y termina en %08X\n",work[i].start,
work[i].end );
   work[i].state = STATE_INVALID;
   sent++;

 if ( __use_http )
 {
  v3HTTPClose( &http );
 }
 else
 {
  v3NetClose( &net );
 }
 return( sent );
}
```

3.6.5.3 Conclusión:

El objeto claro de esta plataforma es el de poder ofrecer una capacidad de distribución de trabajo en diferentes máquinas, pero sin dejar de lado la facilidad del objetivo. Con tan sólo unos ligeros conocimientos del resto de funciones de la plataforma y una buena utilización de las que se han presentado en el apartado anterior, se puede conseguir, con más o menos complicación dependiendo de lo que se quiera desarrollar, una plataforma Cliente/Servidor que sea capaz de dividir la carga de trabajo entre los diferentes clientes, con la consiguiente reducción de costes que implicaría el tener una única máquina más potente para realizar el trabajo.

Como resumen a este apartado, se puede comentar que con unos ligeros conocimientos del resto de las funciones de la plataforma y con unas pequeñas modificaciones de las

funciones anteriormente comentadas, se podría obtener, de manera rápida y sencilla, una distribución de carga de trabajo para cualquier caso posible.

Capítulo 4

Resultados

ÍNDICE DEL CAPÍTULO 4:

4. RESULTADOS:

4.1 Introducción:

En el apartado de resultados se va a mostrar un ejemplo desarrollado para probar la plataforma Cliente/Servidor.

Con este ejemplo se realizan una serie de pruebas con las que obtendremos unos resultados, los cuales serán estudiados para mostrar todos los puntos de vista positivos que se producen al hacer uso de la distribución de carga de trabajo en diferentes máquinas.

4.2 El Problema de Collatz:

En las ciencias matemáticas existen una gran cantidad de conjeturas y problemas que nunca han sido realmente resueltos. Muchos de ellos referentes a la Teoría de Números, como lo fue durante mucho tiempo el conocido **"Último Teorema de Fermat"**, el cual no fue resuelto hasta finales del siglo XX aún cuando fue propuesto hacia la mitad del siglo 17 (unos 350 años aproximadamente tardó su solución en ver la luz).

Para quien desconozca este Teorema, lo que propone es que la ecuación

$$x^n + y^n = z^n$$

no tiene solución en los enteros positivos si n > 2. Esta proposición fue escrita en el margen de una página de una copia de *Aritmética* de Diofanto, la cual se acompañaba por una nota que decía:

> *"He descubierto una bella demostración de esta fórmula, pero este margen es demasiado angosto para escribirla."*
>
> *Pierre de Fermat*

Se concedía un premio para quien lograra demostrarlo y, curiosamente, los fracasos fueron más fecundos que la demostración, pues dieron lugar al nacimiento de algunas de las ramas de la matemática contemporánea.

Sin embargo, este apartado no pretende discutir conjeturas (o proposiciones, teoremas, o como quiera denominarse) que ya hayan sido victoriosamente vencidas con una demostración de su certeza o con un contraejemplo, sino que lo que se busca es discutir problemas aún no resueltos, como es el caso del problema propuesto por el matemático e informático alemán L. Collatz que, literalmente, reza así:

> *"Dado un número natural m > 0. Si el número es par, se divide por 2. En caso contrario, se realiza la siguiente operación: 3m+1".*

La conjetura fue que, para todo número con esas operaciones, terminaba convergiendo a 1.

Ciertamente, este problema está redactado de una manera bastante *computacional*, pero se puede fácilmente traducir a un lenguaje más *"matemático"*. El problema del que habla Collatz dice lo siguiente: existe una función **F** que genera para cada número natural *m* una secuencia de números naturales de términos $\mathbf{a_k}(m)$ tal que:

$\mathbf{a_1}(m)= m$
$\mathbf{a_{n+1}}(m)=\mathbf{3\cdot a_n}(m) + 1$ si $\mathbf{a_n}(m)$ es impar
$\mathbf{a_{n+1}}(m)=\mathbf{a_n}(m)/2$ si $\mathbf{a_n}(m)$ es par
La secuencia termina en el momento que $\mathbf{a_n}(m)=1$

La conjetura de Collatz viene a decir que no importa qué número sea *m*; la secuencia que genera es finita, es decir, para todo número entero $m > 0$ existe un número natural *n* tal que $\mathbf{a_n}(m)=1$.

Las implicaciones de esta conjetura, aunque no muy importantes, sí son bastante interesantes, pues esto asegura que, en algún momento, la secuencia va a tener una potencia de 2 superior a 2, ya que la única manera de llegar a 1 es venir de 2, para 2 es venir de 4, para 4 es venir de 8, para 8 es venir de 16, para 16 venir de 32 ó de 5 y, a partir de aquí, las posibilidades se acrecentan considerablemente (dos en cada caso de potencia par y una en cada caso de potencia impar).

Ahora, la primera potencia de 2 que aparezca va a ser par, es decir, de la forma $\mathbf{2^{2k}}$, debido a que un término $\mathbf{a_n}(m)$ que sea de la forma $\mathbf{2^k}$ tiene dos posibilidades para su término anterior, que $\mathbf{a_{n-1}}(m)$ haya sido par, en cuyo caso sería también una potencia de dos (sería de la forma $\mathbf{2^{k+1}}$), o bien que fuese impar, en cuyo caso $\mathbf{a_n}(m)$ sería de la forma **3k + 1**, es decir, $\mathbf{a_n}(m)$ = **1 (mód 3)** y, además, **2 = -1 (mód 3)** lo que implica que $\mathbf{2^k = (-1)^k (mód\ 3)}$; por ende, k sería par y la primera potencia de 2 que aparece tiene que ser tambien potencia de 4, potencia par de 2.

Éstas y muchas otras observaciones se han hecho acerca de este tema, pero la verdad es que, en general, sigue siendo una *simple* conjetura a pesar de que ha sido verificada para todos los números naturales hasta $5{,}6\cdot 10^{13}$.

Quizá no sea evidente, pero la cantidad de elementos de las secuencias generadas por un número *m* son relativamente cortas, pues la más larga generada por un número mayor o igual a $\mathbf{10^7}$ es de 686 elementos y es generada por 8400511. Y, además, de una secuencia de 10000 números generados aleatoriamente por un computador de un orden no inferior a 10^{100}, de las secuencias por ellos generadas, la más larga constaba de 3868 elementos generada por un número del orden de 10^{105}.

A continuación, vamos a presentar la serie de números obtenidos desde el 1 hasta el 100; de igual modo, presentaremos la serie más larga generada hasta el momento, la cual fue obtenida a partir de m=8400511 ($7^2 \cdot 17439$).

Datos de las primeras 100 secuencias generadas:

m	Cantidad de números en la secuencia	Secuencia
1	1	1
2	2	2 1
3	8	3 10 5 16 8 4 2 1
4	3	4 2 1
5	6	5 16 8 4 2 1
6	9	6 3 10 5 16 8 4 2 1
7	17	7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
8	4	8 4 2 1
9	20	9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
10	7	10 5 16 8 4 2 1
11	15	11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
12	10	12 6 3 10 5 16 8 4 2 1
13	10	13 40 20 10 5 16 8 4 2 1
14	18	14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
15	18	15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
16	5	16 8 4 2 1
17	13	17 52 26 13 40 20 10 5 16 8 4 2 1
18	21	18 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
19	21	19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
20	8	20 10 5 16 8 4 2 1
21	8	21 64 32 16 8 4 2 1
22	16	22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
23	16	23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
24	11	24 12 6 3 10 5 16 8 4 2 1
25	24	25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
26	11	26 13 40 20 10 5 16 8 4 2 1
27	112	27 82 41 124 62 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1

28	19	28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
29	19	29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
30	19	30 15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
31	107	31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
32	6	32 16 8 4 2 1
33	27	33 100 50 25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
34	14	34 17 52 26 13 40 20 10 5 16 8 4 2 1
35	14	35 106 53 160 80 40 20 10 5 16 8 4 2 1
36	22	36 18 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
37	22	37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
38	22	38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
39	35	39 118 59 178 89 268 134 67 202 101 304 152 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
40	9	40 20 10 5 16 8 4 2 1
41	110	41 124 62 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
42	9	42 21 64 32 16 8 4 2 1
43	30	43 130 65 196 98 49 148 74 37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
44	17	44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
45	17	45 136 68 34 17 52 26 13 40 20 10 5 16 8 4 2 1
46	17	46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
47	105	47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
48	12	48 24 12 6 3 10 5 16 8 4 2 1
49	25	49 148 74 37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
50	25	50 25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
51	25	51 154 77 232 116 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
52	12	52 26 13 40 20 10 5 16 8 4 2 1
53	12	53 160 80 40 20 10 5 16 8 4 2 1

54	113	54 27 82 41 124 62 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
55	113	55 166 83 250 125 376 188 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
56	20	56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
57	33	57 172 86 43 130 65 196 98 49 148 74 37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
58	20	58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
59	33	59 178 89 268 134 67 202 101 304 152 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
60	20	60 30 15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
61	20	61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
62	108	62 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
63	108	63 190 95 286 143 430 215 646 323 970 485 1456 728 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
64	7	64 32 16 8 4 2 1
65	28	65 196 98 49 148 74 37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
66	28	66 33 100 50 25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
67	28	67 202 101 304 152 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
68	15	68 34 17 52 26 13 40 20 10 5 16 8 4 2 1
69	15	69 208 104 52 26 13 40 20 10 5 16 8 4 2 1
70	15	70 35 106 53 160 80 40 20 10 5 16 8 4 2 1

71	103	71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
72	23	72 36 18 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
73	116	73 220 110 55 166 83 250 125 376 188 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
74	23	74 37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
75	15	75 226 113 340 170 85 256 128 64 32 16 8 4 2 1
76	23	76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
77	23	77 232 116 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
78	36	78 39 118 59 178 89 268 134 67 202 101 304 152 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
79	36	79 238 119 358 179 538 269 808 404 202 101 304 152 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
80	10	80 40 20 10 5 16 8 4 2 1
81	23	81 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
82	111	82 41 124 62 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
83	111	83 250 125 376 188 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
84	10	84 42 21 64 32 16 8 4 2 1
85	10	85 256 128 64 32 16 8 4 2 1
86	31	86 43 130 65 196 98 49 148 74 37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
87	31	87 262 131 394 197 592 296 148 74 37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
88	18	88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

89	31	89 268 134 67 202 101 304 152 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
90	18	90 45 136 68 34 17 52 26 13 40 20 10 5 16 8 4 2 1
91	93	91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
92	18	92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
93	18	93 280 140 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
94	106	94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
95	106	95 286 143 430 215 646 323 970 485 1456 728 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
96	13	96 48 24 12 6 3 10 5 16 8 4 2 1
97	119	97 292 146 73 220 110 55 166 83 250 125 376 188 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
98	26	98 49 148 74 37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
99	26	99 298 149 448 224 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
100	26	100 50 25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Secuencia generada más grande hasta 10^7:

$m = 8400511$ ($7^2 \cdot 17439$)
Elementos = 686
Secuencia
8400511 25201534 12600767 37802302 18901151 56703454 28351727 85055182 42527591 127582774 63791387 191374162 95687081 287061244 143530622 71765311 215295934 107647967 322943902 161471951 484415854 242207927 726623782 363311891 1089935674 544967837 1634903512 817451756 408725878 204362939 613088818 306544409 919633228 459816614 229908307 689724922 344862461 1034587384 517293692 258646846 129323423 387970270 193985135 581955406 290977703 872933110 436466555 1309399666 654699833 1964099500 982049750 491024875 1473074626 736537313 2209611940 1104805970 552402985 1657208956 828604478 414302239 1242906718 621453359 1864360078 932180039 2796540118 1398270059 4194810178 2097405089 6292215268 3146107634 1573053817 4719161452 2359580726 1179790363 3539371090 1769685545 5309056636 2654528318 1327264159 3981792478 1990896239 5972688718 2986344359 8959033078 4479516539 13438549618 6719274809 20157824428 10078912214 5039456107 15118368322 7559184161 22677552484 11338776242 5669388121 17008164364 8504082182 4252041091 12756123274 6378061637 19134184912 9567092456 4783546228 2391773114 1195886557 3587659672 1793829836 896914918 448457459 1345372378 672686189 2018058568 1009029284 504514642 252257321 756771964 378385982 189192991 567578974 283789487 851368462 425684231 1277052694 638526347 1915579042 957789521 2873368564 1436684282 718342141 2155026424 1077513212 538756606 269378303 808134910 404067455 1212202366 606101183 1818303550 909151775 2727455326 1363727663 4091182990 2045591495 6136774486 3068387243 9205161730 4602580865 13807742596 6903871298 3451935649 10355806948 5177903474 2588951737 7766855212 3883427606 1941713803 5825141410 2912570705 8737712116 4368856058 2184428029 6553284088 3276642044 1638321022 819160511 2457481534 1228740767 3686222302 1843111151 5529333454 2764666727 8294000182 4147000091 12441000274 6220500137 18661500412 9330750206 4665375103 13996125310 6998062655 20994187966 10497093983 31491281950 15745640975 47236922926 23618461463 70855384390 35427692195 106283076586 53141538293 159424614880 79712307440 39856153720 19928076860 9964038430 4982019215 14946057646 7473028823 22419086470 11209543235 33628629706 16814314853 50442944560 25221472280 12610736140 6305368070 3152684035 9458052106 4729026053 14187078160 7093539080 3546769540 1773384770 886692385 2660077156 1330038578 665019289 1995057868 997528934 498764467 1496293402 748146701 2244440104 1122220052 561110026 280555013 841665040 420832520 210416260 105208130 52604065 157812196 78906098 39453049 118359148 59179574 29589787 88769362 44384681 133154044 66577022 33288511 99865534 49932767 149798302 74899151 224697454 112348727 337046182 168523091 505569274 252784637 758353912 379176956 189588478 94794239 284382718 142191359 426574078 213287039 639861118 319930559 959791678 479895839 1439687518 719843759 2159531278 1079765639 3239296918 1619648459 4858945378 2429472689 7288418068 3644209034 1822104517 5466313552 2733156776 1366578388 683289194 341644597 1024933792 512466896 256233448 128116724 64058362 32029181 96087544 48043772 24021886 12010943 36032830 18016415 54049246 27024623 81073870 40536935 121610806 60805403 182416210 91208105 273624316 136812158 68406079 205218238 102609119 307827358 153913679 461741038 230870519 692611558 346305779 1038917338 519458669 1558376008 779188004 389594002 194797001 584391004 292195502 146097751 438293254 219146627 657439882 328719941

986159824 493079912 246539956 123269978 61634989 184904968 92452484 46226242 23113121 69339364 34669682 17334841 52004524 26002262 13001131 39003394 19501697 58505092 29252546 14626273 43878820 21939410 10969705 32909116 16454558 8227279 24681838 12340919 37022758 18511379 55534138 27767069 83301208 41650604 20825302 10412651 31237954 15618977 46856932 23428466 11714233 35142700 17571350 8785675 26357026 13178513 39535540 19767770 9883885 29651656 14825828 7412914 3706457 11119372 5559686 2779843 8339530 4169765 12509296 6254648 3127324 1563662 781831 2345494 1172747 3518242 1759121 5277364 2638682 1319341 3958024 1979012 989506 494753 1484260 742130 371065 1113196 556598 278299 834898 417449 1252348 626174 313087 939262 469631 1408894 704447 2113342 1056671 3170014 1585007 4755022 2377511 7132534 3566267 10698802 5349401 16048204 8024102 4012051 12036154 6018077 18054232 9027116 4513558 2256779 6770338 3385169 10155508 5077754 2538877 7616632 3808316 1904158 952079 2856238 1428119 4284358 2142179 6426538 3213269 9639808 4819904 2409952 1204976 602488 301244 150622 75311 225934 112967 338902 169451 508354 254177 762532 381266 190633 571900 285950 142975 428926 214463 643390 321695 965086 482543 1447630 723815 2171446 1085723 3257170 1628585 4885756 2442878 1221439 3664318 1832159 5496478 2748239 8244718 4122359 12367078 6183539 18550618 9275309 27825928 13912964 6956482 3478241 10434724 5217362 2608681 7826044 3913022 1956511 5869534 2934767 8804302 4402151 13206454 6603227 19809682 9904841 29714524 14857262 7428631 22285894 11142947 33428842 16714421 50143264 25071632 12535816 6267908 3133954 1566977 4700932 2350466 1175233 3525700 1762850 881425 2644276 1322138 661069 1983208 991604 495802 247901 743704 371852 185926 92963 278890 139445 418336 209168 104584 52292 26146 13073 39220 19610 9805 29416 14708 7354 3677 11032 5516 2758 1379 4138 2069 6208 3104 1552 776 388 194 97 292 146 73 220 110 55 166 83 250 125 376 188 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1

4.3 Resultados obtenidos:

En el presente apartado vamos a mostrar los resultados obtenidos al implementar, en la plataforma Cliente/Servidor, el problema de Collatz.

Para ello, se han generado las modificaciones pertinentes tanto en el código perteneciente a la parte del Servidor como a la del Cliente.

Todos los resultados que se van a mostrar a continuación han sido tomados en una red, la cual tiene las siguientes características:

- Red LAN de 10 Mbps.

- Servidor 1:
  - Sistema operativo: Windows 2000 Professional Service Pack 2
  - Microprocesador: Intel Pentium II
  - Frecuencia de microprocesador: 400 Mhz
  - Memoria RAM: 256 MB

- Servidor 2:
  - Sistema operativo: Windows 2000 Professional Service Pack 2
  - Microprocesador: Intel Pentium III Mobile
  - Frecuencia de microprocesador: 1 Ghz
  - Memoria RAM: 256 MB

- Cliente 1:
  - Sistema operativo: Windows 2000 Professional Service Pack 2
  - Microprocesador: Intel Pentium II
  - Frecuencia de microprocesador: 400 Mhz
  - Memoria RAM: 256 MB
  - Características: Esta máquina se usó a la vez como servidor y cliente en una de las pruebas

- Cliente 2:
  - Sistema operativo: Windows NT 4.0 Service Pack 6
  - Microprocesador: Intel Pentium II
  - Frecuencia de microprocesador: 400 Mhz
  - Memoria RAM: 192 MB

- Cliente 3:
  - Sistema operativo: Windows NT 4.0 Service Pack 6
  - Microprocesador: Intel Pentium II
  - Frecuencia de microprocesador: 400 Mhz
  - Memoria RAM: 256 MB

- Cliente 4:
  - Sistema operativo: Windows 98 Second Edition
  - Microprocesador: Intel Pentium III E
  - Frecuencia de microprocesador: 600 Mhz
  - Memoria RAM: 128 MB

- Cliente 5:
  - Sistema operativo: Windows 2000 Professional Service Pack 2
  - Microprocesador: Intel Pentium III Mobile
  - Frecuencia de microprocesador: 1 Ghz
  - Memoria RAM: 256 MB
  - Características: Esta máquina se usó a la vez como servidor y cliente en una de las pruebas

- Cliente 6:
  - Sistema operativo: Windows 2000 Advanced Server Service Pack 2
  - Microprocesador: Intel Pentium III E
  - Frecuencia de microprocesador: 800 Mhz
  - Memoria RAM: 256 MB

- Cliente 7:
  - Sistema operativo: Windows 2000 Advanced Server Service Pack 2
  - Microprocesador: Intel Pentium II
  - Frecuencia de microprocesador: 400 Mhz
  - Memoria RAM: 64 MB

Todos los resultados que se han obtenido proceden de ejecutar la plataforma Cliente/Servidor con el código necesario para poder resolver de manera distribuida el problema de Collatz. Para hacer más rápida la toma de datos, se calculaba el tiempo que se tardaba en resolver el problema de Collatz para los 5000 primeros números naturales.

Se hicieron diversas pruebas con una filosofía común que era la de ir tomando registros de tiempo con un número determinado de clientes, el cual se incrementaba en la toma de datos posterior.

Para tener más campo de visión de lo que implica tener una distribución de trabajo como la que aquí se presenta, se ha hecho uso de diversos equipos, microprocesadores, diferentes capacidades de memoria RAM, diferentes sistemas operativos e incluso se han metido procesos para tener el servidor dedicado a otras tareas además de la de distribuir el trabajo a los diferentes clientes.

Con relación a esto, la metodología seguida por la plataforma para resolver el problema ha sido la siguiente: el Servidor se encargaba de repartir los números a los clientes cuando éstos le hacían una petición al servidor y esos mismos clientes se encargaban de resolver el problema; una vez obtenían la solución, se la remitían al servidor.

Como se podrá comprobar cuando se muestren las estadísticas, a medida que teníamos más clientes conectados al servidor, el tiempo de resolución de los 5000 primeros números era menor. En un momento dado, alcanzábamos el umbral de la máquina servidor, es decir, el servidor llegaba a su tope de conexiones y no se producía una mejora significativa en los tiempos de cálculo por más clientes que se conectaran contra él; al contrario, podría llegar a empeorar los registros de tiempo.

Todo esto lo veremos más claro con las estadísticas que se mostrarán a continuación.

4.3.1 Escenario primero:

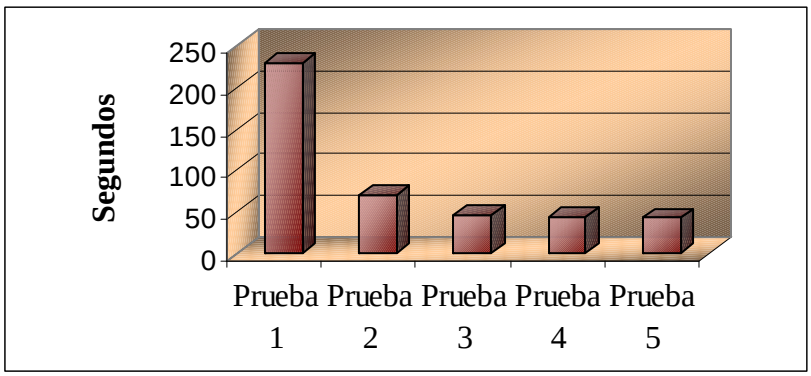
En el siguiente escenario entraron en juego Servidor 1, Cliente 4, Cliente 3, Cliente 6, Cliente 7 y Cliente 5. Este último hacía a la vez de cliente y servidor en una de las pruebas de este escenario.

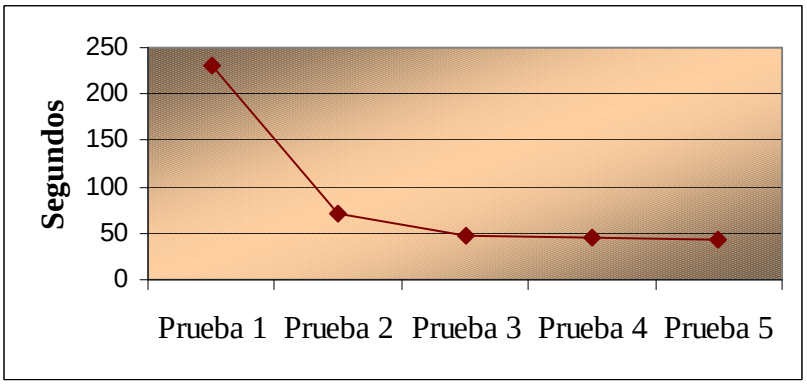
De igual modo, es importante reseñar que la máquina que hacía de servidor, al igual que los clientes, estaban dedicados por completo a la resolución del problema, es decir, no tenían ningún trabajo más que realizar (el microprocesador estaba cediendo el 100% de su capacidad para la prueba).

En la siguiente tabla se mostrarán los resultados obtenidos en el presente escenario:

Número de Prueba	Máquinas que intervienen	Tiempo (segundos)
1	Servidor 1, Cliente 4	230
2	Servidor 1, Cliente 4, Cliente 3	71
3	Servidor 1, Cliente 4, Cliente 3, Cliente 6	47
4	Servidor 1, Cliente 4, Cliente 3, Cliente 6, Cliente 7	45
5	Servidor 1, Cliente 4, Cliente 3, Cliente 6, Cliente 7, Cliente 5	43

A continuación, vamos a mostrar unas gráficas que nos ayudarán a comprender mejor los datos obtenidos:





Como podemos observar en las gráficas, se produce una reducción de tiempo considerable cuando se tiene un solo cliente trabajando contra la máquina servidora (Prueba 1) que cuando este número se incrementa a más.

También podemos observar que, con el aumento del número de clientes (hasta tres trabajando contra la máquina servidora (Prueba 3)), las mejoras de tiempo son considerables y, a partir de ese momento, con más de tres clientes las mejoras de tiempo se siguen produciendo pero ya no son tan pronunciadas. Esto es debido a que la máquina servidora se va acercando al umbral en el que, por más clientes que se conecten, no tiene capacidad para atender de manera tan eficaz las peticiones que se le realizan, repercutiendo así en los tiempos obtenidos en las pruebas.

Es importante indicar que la única máquina que se encontraba conectada contra el servidor en la primera prueba, Cliente 4, es una máquina que tiene su disco duro compartido para toda la red, por lo cual la máquina estaría, además de procesando los datos que recibía del servidor, atendiendo las peticiones que le pudieran llegar de los distintos puntos de la red en la que estaba integrada. Por esta razón, se nota un cambio importante al meter un segundo cliente contra el servidor, puesto que este segundo cliente sí que aumentaba la capacidad de proceso ya que era una máquina dedicada en exclusiva a esta labor, como le ocurría a la máquina que hacía de servidor en este escenario.

4.3.2 Escenario segundo:

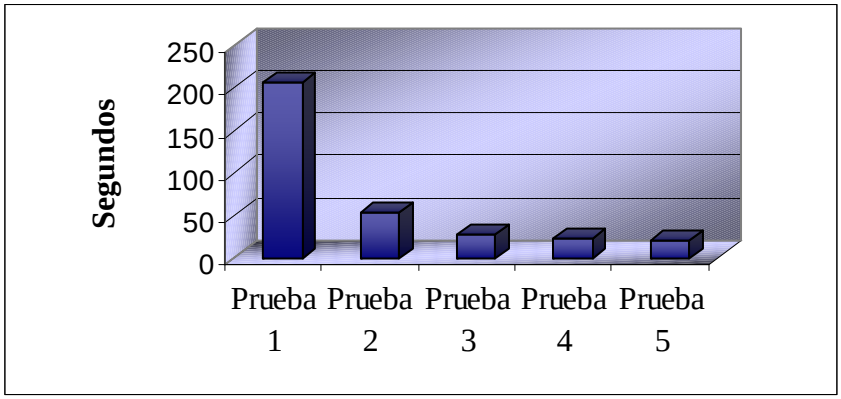
En el siguiente escenario, entraron en juego Servidor 2, Cliente 4, Cliente 3, Cliente 6, Cliente 7 y Cliente 1.

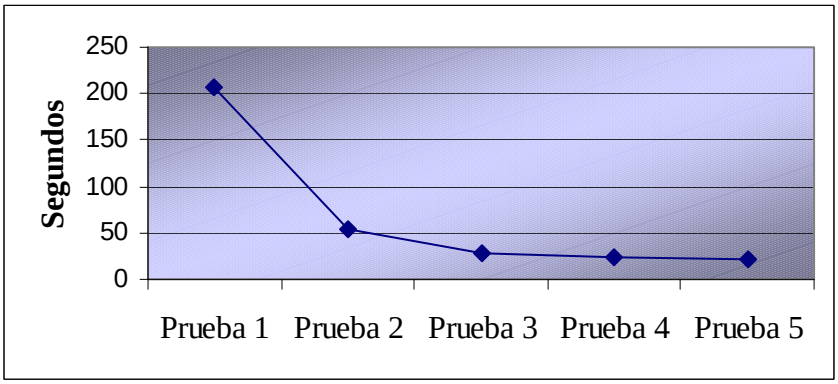
Como ocurría en el escenario anterior, la máquina servidora, al igual que los clientes, estaban dedicados por completo a la resolución del problema, es decir, no tenían ningún trabajo más que realizar (el microprocesador estaba cediendo el 100% de su capacidad para la prueba).

En la siguiente tabla se mostrarán los resultados obtenidos en el presente escenario:

Número de Prueba	Máquinas que intervienen	Tiempo (segundos)
1	Servidor 2, Cliente 4	206
2	Servidor 2, Cliente 4, Cliente 3	53
3	Servidor 2, Cliente 4, Cliente 3, Cliente 6	27
4	Servidor 2, Cliente 4, Cliente 3, Cliente 6, Cliente 7	24
5	Servidor 2, Cliente 4, Cliente 3, Cliente 6, Cliente 7, Cliente 1	21

A continuación, vamos a mostrar unas gráficas que nos ayudarán a comprender mejor los datos obtenidos:





En los resultados obtenidos en este segundo escenario, nos podemos dar cuenta que ocurre una situación similar a la del escenario anterior. Vemos la reducción tan importante que se produce de tener un único cliente conectado con la máquina servidora que tener varios clientes conectados.

Probablemente, esta reducción no sería tan pronunciada si el Disco Duro de Cliente 4 hubiese estado aislado de la red y no hubiese sido una unidad de información compartida por la red LAN en la que se encontraba.

Esta prueba tiene su sentido ya que así, con los resultados obtenidos, podemos mostrar que, a pesar de la poca capacidad de cálculo que nos puedan ceder los clientes que se conectan, siempre se producen mejoras en los tiempos de cálculo ayudándonos así a resolver, de forma más rápida, el problema que nos ocupa.

El umbral de la máquina servidora de este escenario es bastante elevado, puesto que con 5 máquinas clientes conectadas contra ella se producen mejoras de tiempo considerables, lo que nos indica que cuanta más capacidad de máquina tengamos en nuestro servidor, mejores registros de tiempo obtendremos.

4.3.3 Escenario tercero:

En el siguiente escenario entraron en juego Servidor 1, Cliente 2, Cliente 3, Cliente 4, Cliente 6 y Cliente 1.

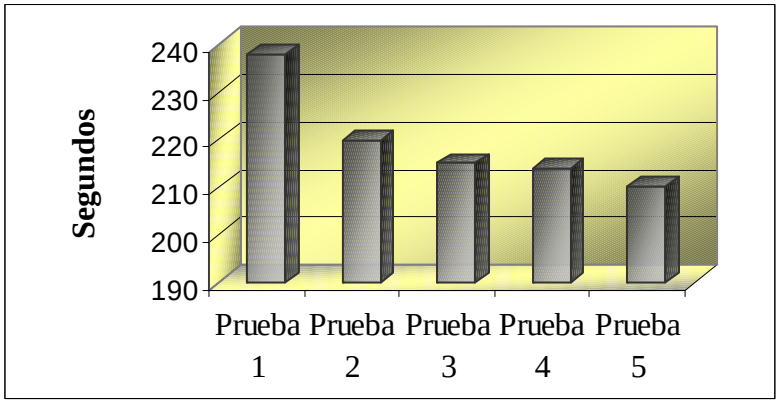
A diferencia de los escenarios anteriores, en éste hemos introducido un pequeño uso de CPU para que, tanto los clientes como la máquina servidora, no estuvieran dedicados por entero a las pruebas desarrolladas. Esta prueba se desarrolló así para poder acercarnos más a lo que ocurre en la vida real, puesto que es muy difícil que los clientes que nos cedan su CPU de manera altruista no estén haciendo uso de sus equipos para sus propios menesteres.

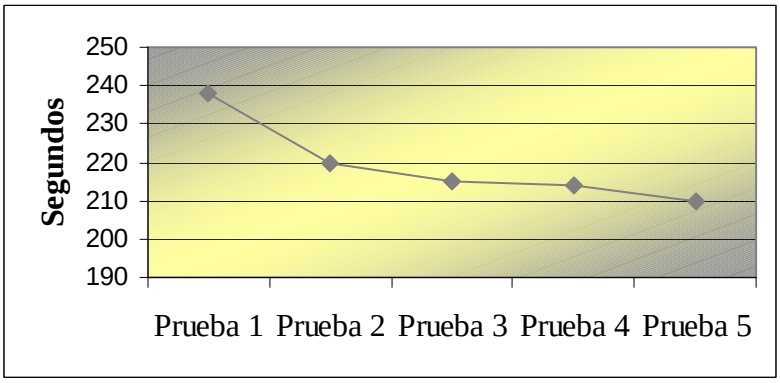
De distinto modo ocurriría con la máquina servidora, puesto que todo servidor, para que lleve a cabo su labor, es aconsejable tenerlo dedicado por entero a la función para la que fue diseñado, pero no está de más comprobar la influencia que podría ejercer el tener, además de la distribución de trabajo entre los diferentes clientes, otro proceso que le quitase tiempo de uso de CPU, de modo que tuviera que repartirse entre ambas tareas incurriendo así en una disminución del servicio prestado a los clientes, por lo que se vería afectado el tiempo de atención a las distintas peticiones realizadas por parte de los clientes.

En la siguiente tabla se mostrarán los resultados obtenidos en el presente escenario:

Número de Prueba	Máquinas que intervienen	Tiempo (segundos)
1	Servidor 1, Cliente 2	238
2	Servidor 1, Cliente 2, Cliente 3	220
3	Servidor 1, Cliente 2, Cliente 3, Cliente 4	215
4	Servidor 1, Cliente 2, Cliente 3, Cliente 4, Cliente 6	214
5	Servidor 1, Cliente 2, Cliente 3, Cliente 4, Cliente 6, Cliente 1	210

A continuación, vamos a mostrar unas gráficas que nos ayudarán a comprender mejor los datos obtenidos:





Como se puede observar en las gráficas, el tiempo empleado para resolver los 5000 primeros números ha sido más elevado que el de los dos escenarios anteriores, situación bastante comprensible puesto que, tanto los clientes como el servidor, no estaban dedicados al 100% a la resolución de este trabajo.

Por esta razón, podemos pensar que el tener una máquina servidora dedicada por completo a esta labor es esencial para que los tiempos empleados en la resolución del problema no se extiendan en demasía, puesto que no sirve de nada tener muchos clientes conectados ayudando a resolver el trabajo si la máquina servidora no es capaz de procesar las peticiones de trabajo que le están llegando.

Esto último lo podemos observar en los tiempos obtenidos: el servidor tardaba más tiempo en procesar las peticiones que recibía, por lo que los clientes pasaban más tiempo ociosos incurriendo en un aumento de los tiempos.

El umbral de la máquina servidora, en este escenario, es más bajo tal y como lo demuestran las gráficas. Más o menos, a partir del tercer cliente conectado las mejoras en los tiempos no son demasiado considerables, hecho bastante predecible puesto que el servidor estaba procesando varias cosas a la vez.

4.3.4 Escenario cuarto:

En el siguiente escenario entraron en juego Servidor 2, Cliente 1, Cliente 2, Cliente 3, Cliente 4 y Cliente 5.

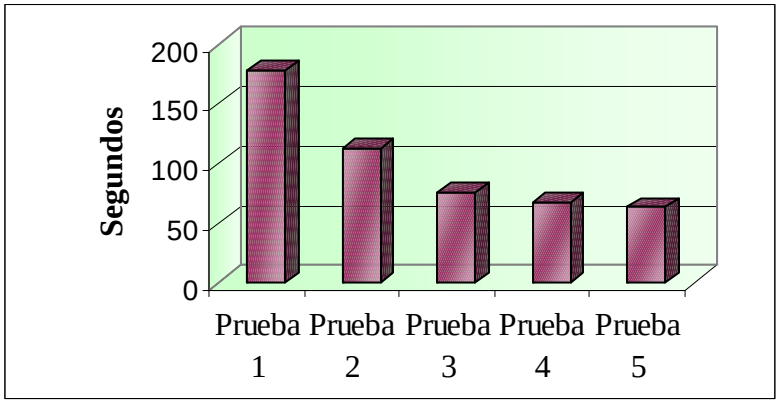
Como en el escenario anterior, se introdujo un pequeño uso de CPU para que tanto los clientes como la máquina servidora no estuvieran dedicados por entero a las pruebas desarrolladas.

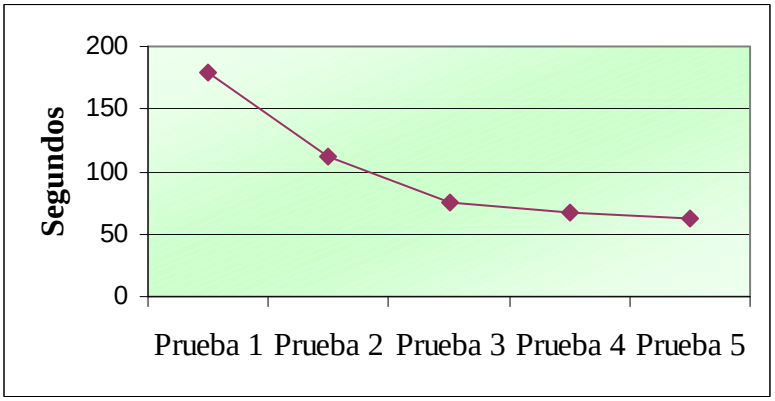
Además de esto, en la última prueba llevada a cabo en este escenario (Prueba número 5), el último cliente conectado al servidor era el propio servidor. Con esta prueba podemos tener dos puntos de vista: uno es el de más carga de trabajo para el servidor, y otro es el del tráfico producido a través de la red puesto que, de este modo, el servidor no tenía que introducir paquetes en la LAN ya que se hacía a través del interfaz de loopback (127.0.0.1).

En la siguiente tabla se mostrarán los resultados obtenidos en el presente escenario:

Número de Prueba	Máquinas que intervienen	Tiempo (segundos)
1	Servidor 1, Cliente 1	179
2	Servidor 1, Cliente 1, Cliente 2	112
3	Servidor 1, Cliente 1, Cliente 2, Cliente 3	75
4	Servidor 1, Cliente 1, Cliente 2, Cliente 3, Cliente 4	67
5	Servidor 1, Cliente 1, Cliente 2, Cliente 3, Cliente 4, Cliente 5	63

A continuación, vamos a mostrar unas gráficas que nos ayudarán a comprender mejor los datos obtenidos:





En estas gráficas podemos observar que las mejoras de tiempo se producen de un modo menos pronunciado que en los escenarios anteriores. Esto es debido a que, al conectarse más clientes, tenemos más capacidad de cálculo pero, al tener otros trabajos que realizar tanto en los clientes como en el servidor, ese aumento de la capacidad de cálculo no es tan importante como en los otros casos.

El umbral de la máquina servidora en este escenario lo podríamos fijar en torno al tercer cliente puesto que, como se puede ver en la segunda gráfica, la mejora de tiempo en ese punto comienza a ser lineal y no tan pronunciada como antes, aunque se podría seguir admitiendo alguna conexión más ya que la tendencia es lineal pero descendente.

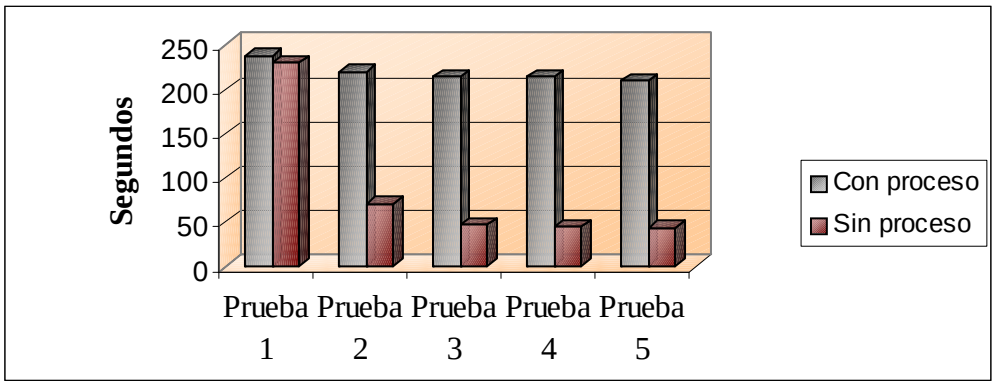
4.3.5 Contraste de escenarios:

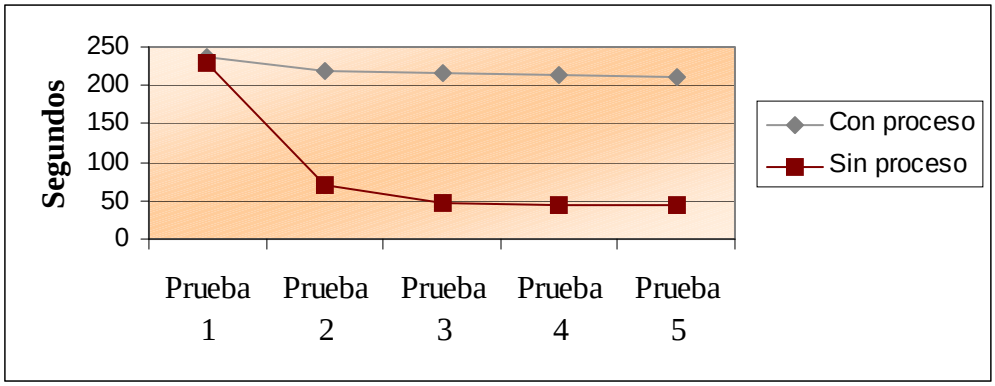
En el presente apartado vamos a realizar una comparativa, desde distintos puntos de vista, acerca de los resultados obtenidos en los diferentes escenarios presentados anteriormente. Los puntos de vista a los que se va a prestar más atención son las diferencias que existen al tener una máquina servidora más potente, la disponibilidad de los clientes que se conectan y la dedicación de los equipos a la resolución del problema; en definitiva, todos esos factores que influyen en el hecho de que tengamos una plataforma de distribución de carga de trabajo lo más eficiente posible.

4.3.5.1 Escenario primero VS. Escenario tercero:

En esta comparativa de resultados, vamos a intentar explicar la diferencia de tiempos obtenidos utilizando como máquina servidora el Servidor 1: en una situación dedicado por entero a la resolución de las peticiones realizadas por parte de los clientes y, en otra, añadiendo un pequeño proceso para dividir la capacidad de proceso del servidor.

Las gráficas comparativas resultantes son las que se presentan a continuación:





Según podemos deducir al observar las gráficas, es un factor muy importante la dedicación que tenga nuestra máquina servidora, puesto que en esta comparativa se nos está mostrando una reducción de tiempos bastante considerable de un caso a otro.

Al principio no se nota demasiado la diferencia de tiempos entre ambos escenarios debido a que el cliente que se encontraba conectado contra el servidor en esa prueba (Prueba 1 en ambos escenarios) era la máquina cuyo disco duro es una unidad de información compartida en la LAN en la que se han llevado a cabo las pruebas, por lo que el cliente, además de procesar los datos que le mandaba el servidor, tenía que atender a las peticiones que le hicieran desde distintos puntos de la red en la que se encontraba conectado.

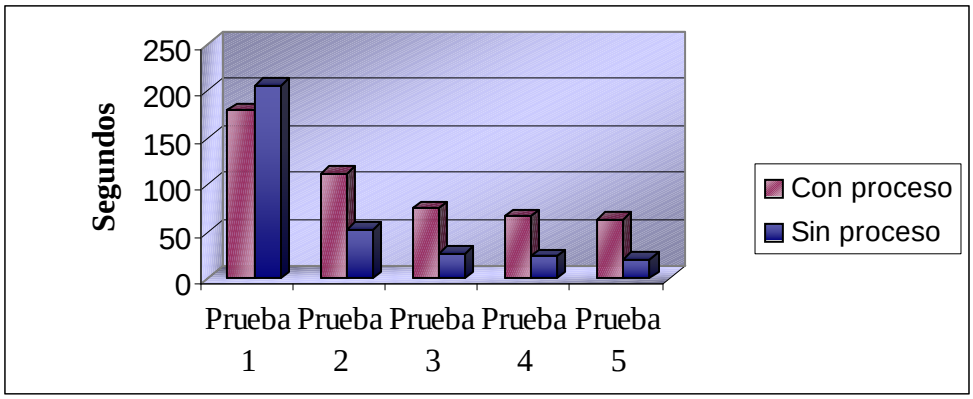
En las sucesivas pruebas sí que se obtiene una mejora considerable en los tiempos. El hecho de tener un servidor dedicado por completo también influye en el umbral de equipos que se pueden conectar para mejorar el rendimiento. En el caso del servidor sin el proceso, es decir, dedicado por completo, el umbral se puede establecer entorno a los tres clientes conectados de manera concurrente, mientras que en la otra situación nos estamos moviendo en torno a los dos clientes; incluso podríamos fijarlo en uno, puesto que la mejora de los tiempos se da de forma bastante lineal, no como en el otro caso que es bastante brusca y descendente.

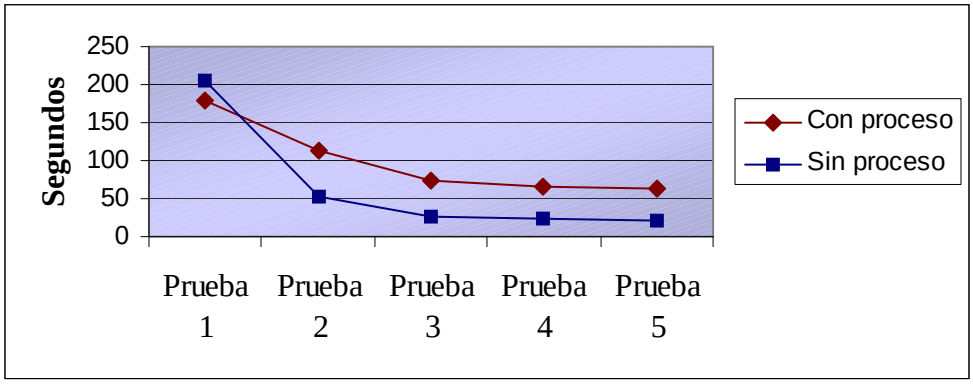
Con respecto a los tiempos obtenidos entre un escenario y otro, la diferencia que se produce es bastante interesante puesto que, a pesar de que al principio (Prueba 1) los tiempos son bastante similares por lo comentado anteriormente, al final entre uno y otro tenemos una diferencia de tiempos entorno a 5 veces menor.

4.3.5.2 Escenario segundo VS. Escenario cuarto:

En esta comparativa de resultados, como ya hiciéramos en la anterior, vamos a enfrentar los resultados obtenidos al usar el Servidor 2: en un caso dedicado por entero a la prueba que se estaba realizando, y en otro repartiendo su capacidad de cálculo con otro proceso.

Las gráficas comparativas resultantes son las que se presentan a continuación:





En la primera prueba podemos ver un factor algo contradictorio ya que, cuando el servidor está dedicado por completo, tarda más tiempo en resolver los 5000 primeros números que en el otro caso. Esta situación es debida a que el cliente que se conecta al servidor es distinto en los dos casos. En el caso en que el servidor no tiene un proceso añadido, es en la máquina a la que hemos venido haciendo referencia hasta ahora la que su disco duro es una unidad de información compartida dentro de la LAN, mientras que en el otro caso es una máquina dedicada por entero a la prueba.

De este modo, vemos que no sólo la dedicación del servidor a la plataforma de distribución es importante, sino que la de los clientes también lo es, aunque en menor medida, puesto que según se iban incorporando conexiones de clientes al servidor el registro de tiempos iba mejorando.

En los tiempos registrados de las pruebas finales se nota una diferencia entorno a 3 veces menor en el caso de tener el servidor dedicado por completo a atender las peticiones de los clientes que el tenerlo dedicado a la vez a otro proceso.

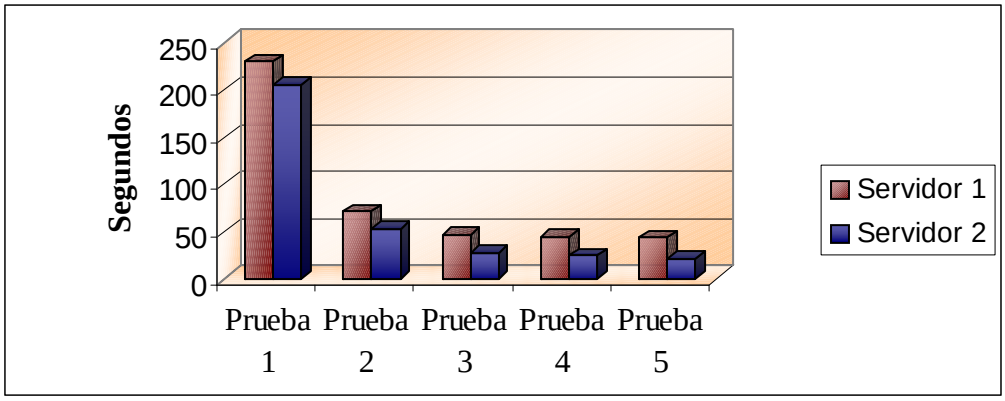
En este caso, el umbral de la máquina servidora en ambas pruebas es similar, en torno a 3 clientes conectados. Del mismo modo podemos observar que los tiempos mejoran en ambas pruebas simétricamente. Esto es debido a que en una prueba tenemos el servidor dedicado por completo, pero existe un cliente (Cliente 4) que realiza otras labores, por lo que la capacidad de cálculo que puede añadir a la plataforma distribuida es menor; en la otra, el que está atendiendo a otro proceso es el servidor. Como podemos ver, esto último afecta en mayor medida puesto que los tiempos registrados son mayores, algo lógico porque el que lleva “la voz cantante” en la distribución es el servidor.

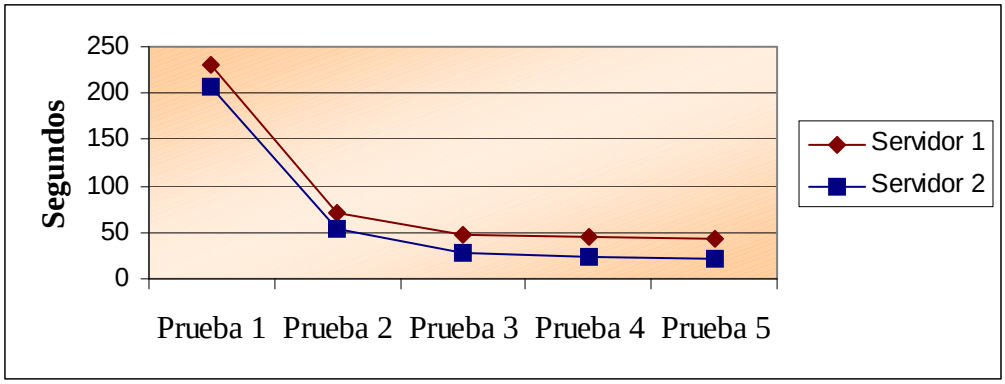
4.3.5.3 Escenario primero VS. Escenario segundo:

En esta comparativa de resultados vamos a presentar las diferencias de tiempos que se producen al tener un servidor con unas características de Hardware más potentes. Previo a los resultados es de suponer que, cuanto más potente sea la máquina servidora, mejores resultados se deben obtener puesto que tendrá mayor capacidad para poder admitir más clientes, es decir, el umbral del servidor será mucho mayor cuantas mejores características posea la máquina.

Es importante comentar que los resultados que se van a comparar hacen referencia a las pruebas realizadas con los servidores dedicados por entero al proceso de distribución de carga de trabajo entre los diferentes clientes conectados, es decir, sin ningún otro proceso adicional que lo ralentice.

Las gráficas comparativas resultantes son las que se presentan a continuación:





Como se puede observar en las gráficas, vemos que se produce una diferencia ostensible entre ambos escenarios y con las mismas características de pruebas; lo único que varía es la máquina que hace de servidor.

A pesar de que las diferencias de hardware no son muy altas, puesto que ambas máquinas tienen 256 MB de RAM, sí que encontramos diferencias en los microprocesadores: el servidor 1 posee un Pentium II 400 Mhz y el servidor 2 posee un Pentium III Mobile 1 Ghz.

Con tan sólo una pequeña modificación en las características del hardware de la máquina servidora, apreciamos que tenemos una reducción de tiempo entorno a la

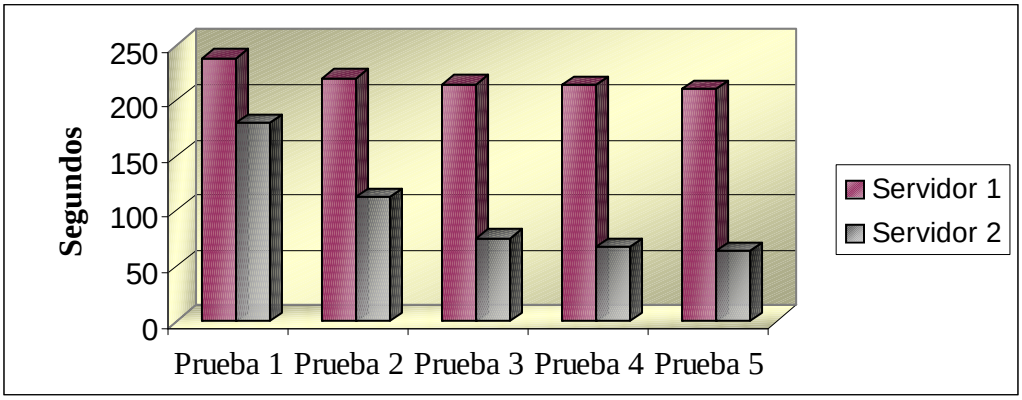
mitad. Con esta premisa, introduciendo un par de mejoras más en la máquina servidora, podríamos obtener una mejora en los registros bastante importante.

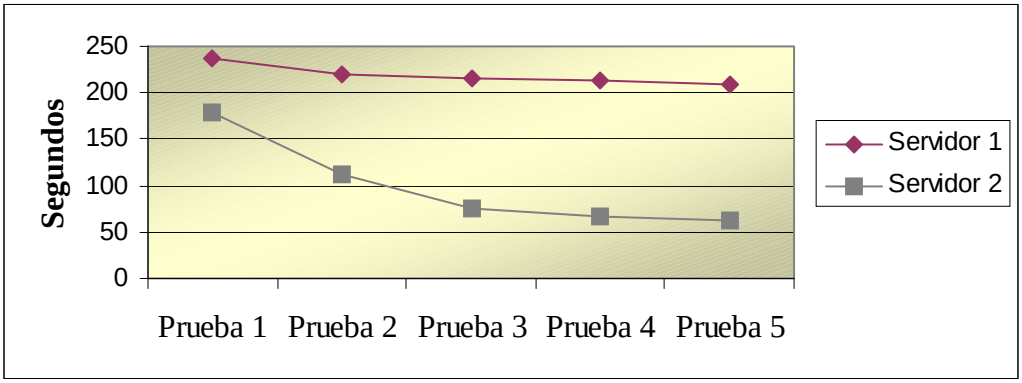
También podemos observar que el umbral que establecen tanto uno como otro servidor es bastante similar debido a que las características, como hemos comentado, no son tan dispares. En ambos casos podríamos fijar este umbral entorno a los 3 clientes conectados de manera concurrente.

4.3.5.4 Escenario tercero VS. Escenario cuarto:

En esta comparativa de resultados vamos a hacer algo más de hincapié en lo comentado en el apartado anterior, añadiendo un proceso a los servidores para que no se dediquen por entero a la atención de los diferentes clientes conectados. Las características de comparación son idénticas que en el caso anterior, con la salvedad del proceso añadido.

Las gráficas comparativas resultantes son las que se presentan a continuación:





De manera similar a lo que sucediera en el apartado anterior, podemos observar que las mejoras de los registros de tiempo son ostensibles entre un caso y otro debido a las diferencias existentes entre ambos servidores. En este caso, si cabe, son más pronunciadas pero por una sencilla razón: teníamos un proceso lanzado para repartir el uso de CPU en los servidores, de manera que la máquina más débil en este aspecto, el servidor 1, se ve más afectada que el servidor 2, que tiene una capacidad de proceso bastante mayor que su homólogo.

Estas diferencias también repercuten más que en el caso anterior, sobretodo en los umbrales de los servidores. En este caso, el umbral del servidor 1 lo podríamos fijar entorno a los dos clientes, mientras que en el servidor 2 estaría cerca de los cuatro clientes.

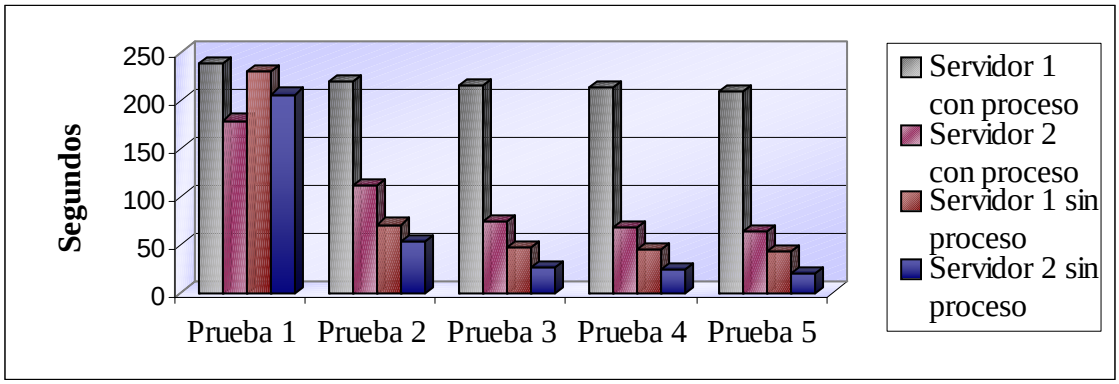
De igual modo, vemos que los tiempos registrados por el servidor 2 son bastante mejores que los obtenidos por el servidor 1: existe una diferencia de casi 4 veces menos en el caso del servidor 2.

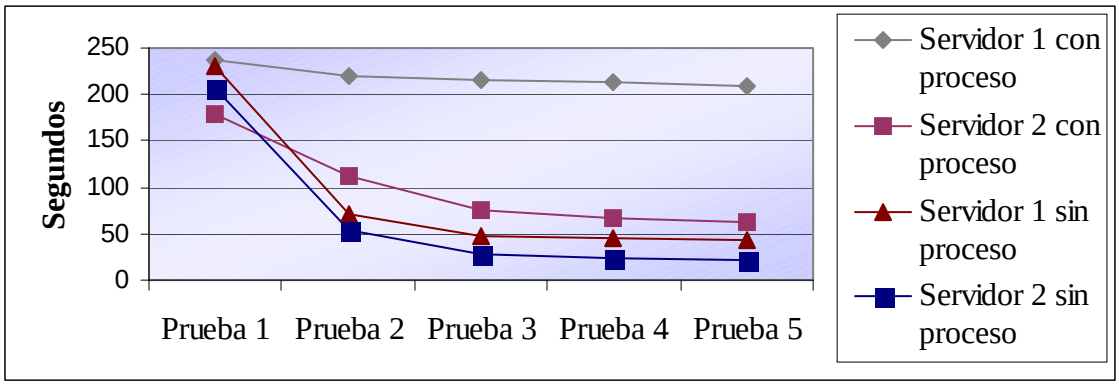
En definitiva, cuanto mejor sea la capacidad de proceso del servidor, mejores resultados obtendremos en cuanto a la celeridad de la distribución y menos afectado resultará si el servidor tiene otras tareas que llevar a cabo.

4.3.5.5 Contraste de todos los escenarios:

En el presente apartado vamos a intentar dar una visión global de todas las pruebas realizadas y de cómo, dependiendo de una serie de parámetros, los resultados que se pueden llegar a obtener puedan ser mejores o peores.

A continuación, vamos a presentar unas gráficas que nos resuman todas las pruebas realizadas para que, con un simple vistazo, nos podamos hacer una idea de los parámetros que estamos buscando:





Como se puede observar en las gráficas, existe una gran diferencia de tiempos entre el servidor 1 y el servidor 2 debido a las diferencias existentes en el hardaware que posee

cada servidor. Esto nos indica que, con una pequeña inversión y una plataforma que sea capaz de distribuir carga de trabajo entre diferentes máquinas, podemos llegar a adquirir una capacidad de cálculo bastante interesante similar a las máquinas con varios procesadores.

De igual modo, estas diferencias también repercuten en el número de clientes conectados concurrentemente que puede llegar a soportar un servidor con ciertas garantías: a más clientes conectados con garantías, más capacidad de cálculo. Y decimos garantías haciendo referencia a que, una vez sobrepasado el umbral que posee cada servidor, los registros de tiempo no se verán mejorados e incluso podrían llegar a empeorarse puesto que el servidor estará recibiendo más peticiones de las que puede cursar con efectividad.

Es importante mencionar que las mejoras que se pueden obtener realizando unas pequeñas modificaciones no sólo provienen de modificaciones directas en el hardware del servidor. También podríamos obtener mejoras en los registros de tiempo si aumentáramos la capacidad de la red. En este entorno de pruebas, la LAN era una red a 10 Mbps. Probablemente, con una red de 100 Mbps, los tiempos registrados en cada uno de los escenarios hubieran sido mejores puesto que el tiempo que pasarían las tramas en la red sería inferior.

Con respecto a los tiempos obtenidos, la diferencia mayor la tenemos en el servidor 1 con un proceso añadido y en el servidor 2 dedicado por entero a la atención de las peticiones de los clientes conectados. Esta diferencia es bastante rotunda puesto que es, más o menos, entorno a 10 veces menor entre un caso y otro.

En este marco de pruebas no repercute en demasía la reducción de tiempos, ya que los registros máximos no son muy elevados; pero, en un marco de cálculo en el que se tarde entorno a las 24 horas en realizar una tarea, el pasar a realizarla en tan sólo 2,4 horas es bastante significativo. Lo mismo ocurre con una tarea que se tarde 12 meses en llevar a cabo conseguir reducirla a tan sólo 1,2 meses.

4.4 Conclusiones:

En este capítulo de estadísticas se ha intentado mostrar, a partir de un ejemplo práctico como es el problema de Collatz, las ventajas que puede tener el poseer una plataforma de estas características y cómo con tan sólo unas pequeñas modificaciones tanto en los servidores como en la calidad de las máquinas clientes que se pretenden conectar podemos adquirir una capacidad de cálculo bastante importante.

Se ha podido observar que, con una pequeña modificación de hardware en un servidor, se han reducido los tiempos de cálculo de manera significativa. Ni qué decir tiene si estas modificaciones hubieran sido mayores: se habría podido lograr una reducción de tiempos aún mayor. De igual modo, se podrían haber obtenido mejoras realizando modificaciones en la electrónica de red, es decir, aumentando la capacidad de la misma y consiguiendo así una reducción en el tiempo de permanencia de las tramas en la red.

Lo más importante de todo esto es que, con una pequeña inversión de dinero, se puede llegar a obtener una gran capacidad de cálculo similar a la de máquinas de varios procesadores pero con un coste totalmente distinto e ínfimo en comparación.

En resumen, con poco esfuerzo y con una plataforma como la que se presenta en este proyecto, se pueden llegar a conseguir unas capacidades de cálculo inmensas y con posibilidades de mejorarlo si se llevan a cabo unas pequeñas modificaciones en el Hardware de la máquina servidora y con la cesión de CPU de los clientes que se conecten contra ella.

Capítulo 5

Conclusiones y líneas futuras de trabajo

ÍNDICE DEL CAPÍTULO 5:

5. CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO:

5.1 Introducción:

Una vez llegados a este punto, podemos decir que ha terminado toda la explicación del trabajo. Ahora es el momento de sacar las conclusiones de cómo se ha realizado el presente proyecto.

En los capítulos previos se ha intentado dar una visión de lo efectivo que puede resultar el tener una plataforma de distribución de carga de trabajo como la presentada aquí; así como lo fácil que puede ser el distribuir cualquier tipo de problema que se nos presente en la plataforma explicada, es decir, que con tan sólo unas pequeñas modificaciones en el código tendremos, de forma rápida y sencilla, el problema en nuestra plataforma Cliente/Servidor de distribución de carga de trabajo sin necesidad de hacer una inversión económica en maquinaria informática potente, puesto que esa potencia la conseguimos gracias a la cesión de CPU que nos otorgan los clientes conectados al servidor.

En este capítulo comentaremos las conclusiones finales que se extraen del desarrollo del proyecto. Las conclusiones se describen por capítulos y se realizarán unas reflexiones generales acerca del proyecto en su conjunto.

Inicialmente se expondrán las posibles mejoras que se podrían realizar en el proyecto junto con las limitaciones del mismo.

Seguidamente pasaremos a comentar las líneas de trabajo por las que debería ir dirigida una ampliación del presente proyecto, u otro que intente modelar algo similar a lo presentado en el actual.

5. 2 Mejoras:

Dentro de este apartado vamos a indicar cuáles serían las mejoras que se podrían llevar a cabo en el presente proyecto.

Una de las más importantes sería la de realizar un IGU (Interfaz gráfico de usuario) puesto que el presente proyecto carece de él y se basa en la ejecución a través de la línea de comandos.

En ese interfaz gráfico que se aconseja desarrollar, podría ser un punto importante el englobar los parámetros a introducir por parte del usuario, como pudiera ser la IP del servidor (si es que estamos hablando del programa cliente), o un botón para borrar la configuración tanto del cliente como del servidor para no tener que poner, en el caso del cliente, c*lient –config* y, en el caso del servidor, *server –config*, en la línea de comandos como se hace actualmente.

Otra de las posibilidades que se le podría otorgar a ese interfaz gráfico sería que tuviese un IDE (Interface Developer Enviroment), es decir, un interfaz de programación con el objetivo de tener claramente diferenciadas las 4 funciones que se tienen que modificar, tanto en la parte del cliente como en la del servidor, para que la plataforma distribuya una determinada carga de trabajo u otra. De esta forma, conseguiríamos de manera más sencilla aún que la plataforma fuera rápidamente modificable y estuviera a disposición, en el tiempo más breve posible, para su explotación.

Un aspecto técnico pero de gran relevancia para la mejora en los tiempos empleados para la resolución del problema sería la selección, por parte del servidor, de todos los clientes que se pueden conectar a él y, llevándolo a un extremo más clasista, qué clientes se pueden conectar a él.

Esta filosofía sería un factor importante para que los tiempos de ejecución que se obtengan sean mejores y que el servidor tuviera conocimiento de cuál es su umbral de máquina en cada momento. Así, podría despreciar conexiones de clientes que le llegaran evitando sobrecargarse y ralentizar el proceso que se está llevando a cabo. Esto se puede realizar de manera rápida y sencilla teniendo un contador que lleve este número de umbral autocalculado en cada instante de tiempo; una vez llegado al umbral, el servidor tan sólo tiene que ignorar los clientes nuevos que vayan llegando, consiguiendo así evitar una sobrecarga de clientes y, por tanto, una pérdida de capacidad de cálculo.

Otro de los factores comentados sería el de saber qué clientes se están intentando conectar al servidor. Esto no implica más que descartar a aquellos clientes que no tuvieran unas características mínimas para mejorar la capacidad de cálculo que se tenga en cada momento. Se ha podido ver en el capítulo anterior que, en una de las pruebas realizadas, existía una máquina que se encontraba como servidor de datos y empeoraba el rendimiento de la plataforma y, por tanto, los tiempos que se obtenían en el cálculo de los 5000 primeros números era mayor que en otros escenarios.

Para poder desarrollar este punto, sería necesario obtener características técnicas en el cliente y remitírselas al servidor antes de conectarse. En este momento, el servidor tendría que ser capaz de discernir si ese cliente va a beneficiar o no a la comunidad de clientes existentes y, dependiendo de lo que ocurriera, dejarle conectarse o no.

La mejora que vamos a pasar a comentar se puede ver desde un punto de vista más subjetivo, es decir, las anteriores son mejoras claras y obvias que se pueden llevar a cabo para mejorar el rendimiento de la plataforma, mientras que la que se va a proceder a comentar no lo es tanto.

Decimos que es subjetiva porque puede tener seguidores y detractores. La mejora va focalizada hacia el tráfico de red que se produce. Centrándonos en el problema llevado a cabo para obtener las estadísticas (el problema de Collatz), el servidor, a cada petición de trabajo que le realizaba un cliente, le mandaba tan sólo un número para procesar; por lo tanto, se generaban gran cantidad de paquetes que inundaban la red pero de reducido tamaño. La mejora, que ya se atisba más o menos, está orientada a mandar más cantidad de trabajo por parte del servidor hacia los clientes, es decir, que el servidor, en vez de mandar sólo un número para que el cliente procese, mande, por ejemplo, 10 números. De este modo, el número de paquetes que circulan por la red se reduce pero, a su vez, el tamaño se multiplica. Por esta razón, se ha comentado la dualidad de interés que puede generar este aspecto: algunas veces nos podrá beneficiar, pero en otras no tanto; de todos modos, eso va de la mano de la persona que esté desarrollando las modificaciones de la plataforma para el problema que se le presente.

Es digno de mención que los problemas para los que está pensada una plataforma de estas características requieren de una capacidad de cálculo bastante importante por parte de los clientes y, si el servidor les asigna varias tareas de cálculo de este estilo a la vez, la mejora de los tiempos se puede ver reducida.

Dado el potente funcionamiento de la plataforma no existen, a priori, nuevas mejoras que se puedan implementar aunque, con la continua evolución de la informática, seguro que en un futuro nos podríamos plantear nuevas modificaciones.

5. 3 Conclusiones:

En este apartado se van a exponer las conclusiones extraídas tanto de los diferentes capítulos que componen la memoria como las conclusiones globales del conjunto del proyecto. Se pretende medir el resultado del trabajo realizado y el cumplimiento de los objetivos originales que fueron la causa de la realización de este proyecto.

5.3.1 Conclusiones globales:

Centrándonos en el punto de vista presentado en el primer capítulo de la presente memoria, existen grupos de personas que están desarrollando continuamente proyectos para ayudar a la comunidad científica en sus estudios. Estos proyectos se realizan de manera desinteresada (como es el caso de éste).

Con esta visión, las empresas se decantarían por una solución de este estilo, es decir, en vez de hacer una inversión económica importante en una máquina que pudiera otorgar una capacidad de cálculo interesante, se haría uso de una plataforma Cliente/Servidor para distribuir la carga de trabajo y la inversión sería menor puesto que se haría en los equipos normales que suelen existir en cualquier empresa, universidad, laboratorio, etc... Incluso si llevamos este punto de vista a Internet, esta inversión sería mucho menor puesto que existen muchos miles de internautas que estarían dispuestos a prestar el uso de su CPU para una labor científica; de hecho, existen proyectos con esta filosofía como el comentado a lo largo de esta memoria, SETI@HOME.

Como se ha podido observar, introducir un problema que lleve asociado un peso de cálculo importante en nuestra plataforma no es difícil puesto que, con tan sólo la modificación de unas pocas funciones, lo tendríamos a disposición de ser explotado de manera rápida y sencilla.

Volviendo al aspecto económico, el tener una plataforma de este estilo, como todo lo relacionado con la informática, con el paso del tiempo se puede quedar obsoleta. Pues bien, si hiciéramos una inversión en una máquina para que llevara todo el peso del cálculo, el impacto en la empresa podría ser elevado, mientras que con una plataforma como la que hemos estado comentando a lo largo de todo este proyecto bastaría con una pequeña inversión en los ordenadores que harían de clientes; además, podríamos mejorar incluyendo clientes, o con unas pequeñas modificaciones en la máquina servidora puesto que, como hemos podido ver en el capítulo de estadísticas, obteníamos unas mejoras considerables con sólo unas pequeñas modificaciones de hardware, cuyo coste es liviano para las empresas o facultades que quisieran hacer uso de esta aplicación.

5.3.2 Conclusiones parciales:

Las conclusiones que se han ido extrayendo a lo largo del desarrollo de los diferentes capítulos de manera independiente se pasarán a mostrar a continuación:

Capítulo 1. En este capítulo se ha expuesto el motivo por el cual se ha realizado el presente proyecto. Asimismo, se han sentado las bases y los objetivos que persigue la plataforma Cliente/Servidor para la distribución de trabajo. Por tanto, podemos comentar que las conclusiones que se obtienen de este capítulo son la necesidad de resolver el problema que se le puede platear a una empresa, facultad, etc.., de tener que invertir en una máquina potente para poder resolver un determinado problema que exija una capacidad de cálculo importante: el problema queda resuelto mediante el uso de una plataforma como la desarrollada y la posibilidad de llevarlo a buen puerto especificando tanto objetivos como el problema a atajar.

Capítulo 2. En este capítulo se han comentado las distintas arquitecturas existentes para la comunicación de un sistema distribuido. Del mismo modo, se han expresado las diferentes elecciones existentes para la comunicación entre los clientes y servidores. Una buena elección en el sistema a utilizar será un punto clave para el desarrollo del cualquier proyecto en el que se vea implicado un sistema que cumpla el paradigma del Cliente/Servidor.

Capítulo 3. En este capítulo se ha tratado de presentar el objeto claro de la plataforma, el poder ofrecer una capacidad de distribución de trabajo en diferentes máquinas pero sin dejar de lado la facilidad del objetivo. Con tan sólo unos ligeros conocimientos del resto de funciones de la plataforma y una buena utilización de las mismas, se puede conseguir, con más o menos complicación dependiendo de lo que se quiera desarrollar, una plataforma Cliente/Servidor que sea capaz de dividir la carga de trabajo entre los diferentes clientes, con la consiguiente reducción de costes que implicaría el tener una única máquina más potente para realizar el trabajo.

Capítulo 4. En este capítulo de estadísticas se ha intentado mostrar al lector, a partir de un ejemplo práctico como es el problema de Collatz, las ventajas que puede tener el poseer una plataforma de estas características, y cómo con sólo unas pequeñas modificaciones tanto en los servidores como en la calidad de las máquinas clientes que se pretenden conectar podemos adquirir una capacidad de cálculo bastante importante. Hemos podido observar que, con tan sólo una pequeña modificación de hardware en un servidor, se han reducido los tiempos de cálculo en bastante medida. Ni qué decir tiene lo que se habría logrado si estas modificaciones hubieran sido mayores: una reducción de tiempos mucho mejores. De igual modo, se podrían haber obtenido mejoras realizando modificaciones en la electrónica de red, es decir, aumentando la capacidad de la misma. Lo más importante de todo esto es que, con una pequeña inversión de dinero, se puede llegar a obtener una gran capacidad de cálculo, similar a la de máquinas de varios procesadores, pero con un coste totalmente distinto e ínfimo en comparación. En resumen, con poco esfuerzo y con una plataforma como la que se presenta en este proyecto se pueden llegar a conseguir unas capacidades de cálculo inmensas y con posibilidades de mejorarlo con tan sólo unas pequeñas

modificaciones en el hardware de la máquina servidora y con la cesión de CPU de los clientes que se conecten contra el servidor.

5.4 Líneas futuras de trabajo:

Las líneas futuras de trabajo pueden tener una fuerte base en el proyecto que aquí se ha expuesto. No obstante, existen varios puntos que pueden ser replanteados o tratarse nuevas funcionalidades para el sistema.

A lo largo de toda la memoria, se han indicado los apartados que pueden ser mejorados y sobre los que hay que poner interés en el futuro, pero un resumen de los pasos que son susceptibles de seguir en posteriores trabajos se encuentra en el apartado de mejoras del presente capítulo (**5.2 Mejoras**).

Uno de los puntos fuertes de la herramienta es la portabilidad que tiene hacia la mayoría de los sistemas operativos existentes en la actualidad, como por ejemplo Windows (plataforma Win32-x86-msvc), Linux (plataforma x86, sparc64, sparc, ppc, alpha-ccc, alpha), Solaris (plataforma sparc, sparc64, sparc64-workshop, sparc-workshop, x86), Sunos (plataforma sparc), etc.

Por lo tanto, una de las líneas que habría que seguir en el futuro sería la de mantener la plataforma actualizada para las versiones de los sistemas operativos que pueden surgir, en los cuales hoy es portable la plataforma y además adecuarla a aquellos sistemas operativos potentes en el mercado actual como por ejemplo AIX, THEOS, etc, para los que aún no es portable.

Recordando una de las mejoras a realizar en el apartado 5.2 del presente capítulo, y debido a las tendencias que se llevan en la actualidad acerca del aspecto de las herramientas informáticas, uno de los posibles y más aceptados interfaces gráficos por parte de los usuarios sería un entorno web, es decir, una herramienta con aspecto de página web puesto que, actualmente, el software en alguno de sus apartados o módulos tiene ese aspecto, ya que el “Look & Feel” de las páginas web es más familiar y manejable para los usuarios, por lo que podría tener más aceptación.

En una sociedad como la actual en la que se busca lo más potente y lo más rápido en lo que a tecnología se refiere, hay que hacer mención a otra de las mejoras comentadas con anterioridad: la capacidad que podría tener el servidor de determinar qué clientes se conectan y cuántos se pueden conectar a él en un momento determinado. Este aspecto debe marcar una clara línea de trabajo en el futuro puesto que, de este modo, conseguiremos el mejor rendimiento de la plataforma en cada momento, logrando así los valores mencionados al principio de este comentario.

Otra línea que se antoja importante para que esta plataforma tenga aceptación en la comunidad científica es la de montar un portal web donde las personas interesadas puedan hacer uso del código desarrollado y realizar preguntas acerca de las posibilidades que podría otorgarles nuestro sistema para sus estudios. Del mismo modo, también intentar resolverles todas las dudas técnicas que se les pudieran presentar a la hora de adecuar la plataforma a sus intereses personales.

Bibliografía

BIBLIOGRAFÍA:

Artículos y publicaciones:

📖 Kay A. Robbins, Steven Robbins. “Unix programación práctica. Guía para la concurrencia, la comunicación y los multihilos”. Prentice Hall.

📖 Céderic Nicolás, Cristophe Avare, Fréderic Najmar. “Java Cliente Servidor”. Eyrolles.

📖 Félix García Carballeira. “Introducción al Lenguaje de Programación C”, 1999.

📖 Andrew S. Tanenbaum. “Distributed operating systems”, 1995. Prentice Hall International.

📖 Sape Mullender. “Distributed Systems”, 1993. Second Edition. Addison-Wesley.

📖 P.H. Enslow. “What is a Distributed Data Processing System?”, 1978. IEEE Computer.

📖 G. Coulouris, J. Dollimore, T. Kindberg. “Distributed Systems: Concepts and Desing”, 2001. Third Edition. Addison-Wesley.

📖 A. Tanenbaum, M. Van Steen. “Distributed Systems: Principles and Paradigms”, 2002. Prentice Hall.

📖 Pradeep Sinha. “Distributed Operating Systems”, 1997. IEEE Press.

📖 Bergman C.M., Sawley M.L. “The use of the message passing model on distributed networks for Computational Fluid Dynamics”. EPFL Supercomputing Review, Nº 5.

📖 Cook, Pancake and Walpole. “Are expectations for parallelism too high? A survey of potential parallel users”, 1994. IEEE Trans. on Computers.

📖 Germund Dahlquist, Ake Björck. “Numerical Methods”, 1974. Prentice Hall.

📖 Eager, Lazowska and Zahorjan. “Adaptative Load Sharing in Homogeneous Distributed Systems”, 1986. IEEE Trans. on Soft. Eng.

📖 Krueger and Livny. “The diverse Objetives of Distributed Scheduling Policies”, 1987. IEEE Computer.

📖 López A. “Procesamiento distribuído en una red de workstations heterogéneas”, 1993. 21ª Conferencia Latinoamericana de informática.

📖 Pablo Prato, Álvaro Fernández, Antonio López. “Estudio Comparativo de Diferentes Sistemas Operativos para el Desarrollo de Aplicaciones Distribuidas”, 1998.

🕮 Shivaratri, Krueger and Singhal. “Load Distributing for Locally Distributed Systems”, 1992. IEEE-Computer.

🕮 Waldspurger, Hogg, Hunerman, Kephart, Stornetta, Spawn. “A Distributed Computational Economy”, 1992. IEEE Trans on Soft.

🕮 Wang and Morris. “Load Sharing in Distributed Systems”, 1985. IEEE Trans on Computers.

🕮 James Cooper. “Computer and Communications Security”, 1989. Mc. Graw Hill.

🕮 E. Dawson and J. Golic. “Criptography:Policy and Algoritms”, 1995. Spriger Verlag.

🕮 Leonard M. Adleman, Ming-Deh Huang. “Algoritmic Number Theory”, 1994. Springer-Verlag.

🕮 G.H. Hardy, E.M. Wright. “An introduction to the Theory of Numbers”, 1993. Oxford.

🕮 N. Koblitz. “A course in number theory and cryptography”, 1987. Springer-Verlag.

🕮 Paulo Ribenboim. “The little book of BIG primes”, 1991. Springer-Verlag.

Proyectos Fin de Carrera:

🕮 Laura Díaz García. “Técnicas de acceso a sistemas de información basadas en JDBL”. Universidad Carlos III de Madrid.

🕮 Ángel Serrano Dorado. “Informatización de la gestión de albergues del Camino de Santiago”. Junio de 2000. Universidad Carlos III de Madrid.

🕮 Christian Díaz Pérez. “Análisis y desarrollo de aplicaciones en Active Server Pages para entornos Internet, Intranet y Extranet.” Febrero de 2002. Universidad Carlos III de Madrid.

Páginas Web:

- http://www.mithral.com
- http://www.gnutella .com
- http://www.linux.org
- http://www.cisco.com
- http://www.microsoft.com
- http://www.dcc.uchile.cl/~lmateu/CC10A/Apuntes/rec/
- http://setiathome.astroseti.org/project.html
- http://www.clearinghouse.gub.uy/~carlos/info-csic97/info_csic97.html

- http://www.fing.edu.uy/cecal/hpc
- http://www.netlib.org/pvm3/book/pvm-book.html
- http://agamenon.unidades.edu.co/sistemas/tema.htm
- http://agamenon.unidades.edu.co/sistemas/6606.htm
- http://agamenon.unidades.edu.co/sistemas/6605.htm
- http://www.inf.utfsm.cl/~rmonge/sd/apunte01.pdf
- http://ar.geocities.com/r_niella/Document/t_cap1.htm
- http://www.inei.gob.pe/biblioineipub/bancopub/Inf/Lib5098/indice.htm