UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

UNIDAD ACADÉMICA DE LOS CICLOS PROFESIONALES Y DE POSGRADO
DEL COLEGIO DE CIENCIAS Y HUMANIDADES

La Estructura de Servidores Distribuidos Para la Adquisición de Datos y Registro de Eventos en Tiempo Real

T E S I S
Que para obtener el título de

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

Presenta el

Lic. José Salvador Villarreal Rodríguez

Asesor : Dr. Mario Albarrán Figueroa
Asesor Nacional : Dra. Hanna Oktaba

1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Dedicatoria

A mis padres : Beatriz Hilda y Salvador.

A mis hermanos : Luis Felipe, Beatriz María, César Emilio, Hernán Manuel y Elena Hilda.

Para mi novia Elizabeth.

AGRADECIMIENTOS

A Dios, por todo.

A mi asesor Dr. Mario Albarrán Figueroa, por sus consejos y su amistad.

A la Universidad Nacional Autónoma de México, especialmente a los profesores y personal administrativo de la Maestría en Ciencias de la Computación, los investigadores del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, por su apoyo en mi paso por la Universidad, especialmente a la Dra. Hanna Oktaba.

A la Universidad Estatal de California, donde realicé el presente trabajo, en particular a los directivos del Departamento de Matemáticas y Ciencias de la Computación, Dr. Veril Phillilps y Dra. Eloise Hamann.

Al Instituto de Investigaciones Eléctricas, por el interés mostrado en la realización del presente trabajo, al Dr. Alejandro Villavicencio Ramírez, al Dr. Roberto Canales Ruíz, y a mis compañeros de trabajo.

Al Consejo Nacional de Ciencia y Tecnología, que me otorgó la beca con la que realicé los créditos de la Maestría.

CONTENIDO

I INTRODUCCIÓN

Desde los inicios de las aplicaciones industriales de la computación, se han desarrollado sistemas para el monitoreo y control de los procesos industriales. Las características de tales sistemas se pueden resumir en las siguientes :

a) Necesitan reaccionar en tiempo real a los eventos del medio ambiente, de tal manera que no pierdan detalle de los eventos que suceden.

b) Es necesario que exista una robustez en el sistema, de tal manera que las fallas de software o hardware afecten en lo mínimo las funciones a realizar. Esta robustez frecuentemente se logra incluyendo elementos redundantes.

Los sistemas distribuidos en tiempo real permiten que se cumpla con los requisitos de tiempo de respuesta, dividiendo el trabajo a realizar en los diferentes elementos, y creando las redundancias necesarias para evitar pérdidas de información ocasionadas por fallas.

El presente trabajo no trata la recuperación de fallas (Fault Tolerance) pero si plantea y evalúa un esquema acceso a una función por medio de un servidor distribuido.

El servidor distribuido permite que una función localizada en un nodo del sistema se pueda accesar concurrentemente por el resto de los nodos. Al repetir la función en dos o mas nodos, permite que si uno falla se pueda accesar la misma función desde otros nodos. Esto último se puede hacer en todos los nodos y de esta forma permitir una redundancia total, lo cual no siempre es deseable, debido a posibles problemas de espacio.

Objetivos

El objetivo principal de este trabajo es probar la validez de la siguiente hipótesis :

En un Sistema de Adquisición de Datos y Registro de Eventos en Tiempo Real, con una arquitectura distribuida donde estaciones de trabajo se conecten por medio de una red local, es posible que algunas funciones se puedan repartir a través de un servidor distribuido, bajo el esquema de cliente-servidor, utilizando llamadas a procedimientos remotos en la comunicación y procesos de peso ligero para permitir la concurrencia de varios clientes y satisfaciendo los requerimientos de tiempo real.

Para evaluar la hipótesis anterior, se han trazado los siguientes objetivos secundarios :

a) En el campo de los Sistemas Distribuidos en Tiempo Real, revisar las áreas de intercomunicación de tareas y de manejo de paralelismo, para evaluar la actuación de diferentes soluciones, en especial la de servidores distribuidos con múltiples procesos de peso ligero (LWPs), que atienden a los clientes que requieren servicio mediante llamadas a procedimientos remotos (RPCs).

b) Conjuntar las metodologías y mecanismos utilizados en servidores distribuidos en diferentes sistemas y por diferentes proveedores, destacando aquellos que se apeguen a normas.

c) Desarrollar un producto de software que sirva de modelo para la construcción de servidores distribuidos para Sistemas Distribuidos en Tiempo Real. El producto consistirá en un servidor distribuido para la función de Registro Histórico de un Sistema de Adquisición de Datos y Registro de Eventos.

Antecedentes

El presente trabajo muestra un estudio de los servidores distribuidos para sistemas en tiempo real, así como su aplicación en Sistemas de Adquisición de Datos y Registro de Eventos (SADRE) para centrales generadoras de energía eléctrica.

Los primeros sistemas SADRE se realizaron en arquitecturas centralizadas con CPUs muy rápidos, que permitían cumplir con las restricciones de tiempo de las tareas. Actualmente se considera que las plataformas para Sistemas en Tiempo Real deban ser, además de rápidas, predecibles, y que es posible que un sistema distribuido sea predecible y de esa forma poder implantar un Sistema Distribuido en Tiempo Real (SDTR).

Existen sistemas comerciales que toman ventaja de lo anterior, y utilizando redes locales (LANs) distribuyen las funciones de adquisición y/o de Interfase Hombre-Máquina (IHM), como ProVox de Fisher, TDC-2000, TDC-3000 de Hewlett Packard, y Network 90 de Bailey. La idea de integrar equipo y software de otros fabricantes está siendo común, así como el concepto de extender las potencialidades de los sistemas agregando nodos, software o interfases con otros sistemas.

En México el Instituto de Investigaciones Eléctricas ha desarrollado sistemas SADRE que cumplen con requisitos específicos de las plantas de generación de energía eléctrica del país, con arquitecturas centralizadas como el SADRE de Tula y el SADRE de Manzanillo, otros como el Sistema de Información para el Registro y Análisis de Transitorios de Laguna Verde utiliza LAN para comunicar el computador central con las unidades de adquisición y con las terminales, también se ha desarrollado el Sistema Integral de Información del Proceso de Laguna Verde, que bajo una red Ethernet tiene distribuida (replicada) la función de IHM, pero está atado en cuanto a las plataformas de software y hardware (VAX/VMS, DECnet, DECwindows ...). Es de particular importancia el hecho de que los SDTR mas atractivos permiten extenderse en sus funciones y potencialidades, además de aceptar plataformas heterogéneas debido al seguimiento de normas en sus desarrollos. El contexto de estudio del presente trabajo considera la tendencia a regirse por normas como UNIX, OSI y TCP/P, X Window System, la norma POSIX para Sistemas Operativos Portables, y la norma "de facto" SVR4 de AT&T.

Resumen

Se presenta el desarrollo y evaluación de un servidor distribuido de la función de registro histórico de un sistema de adquisición de datos y registro de eventos en tiempo real para plantas generadoras de energía eléctrica.

La función se realizó bajo el esquema cliente/servidor, utilizando el paradigma de llamadas a procedimientos remotos en la comunicación, así como el de procesos de peso ligero en la concurrencia de clientes.

Actualmente están emergiendo estándares sobre la tecnología que implementa los paradigmas anteriores y el desarrollo se apego a tales estándares en lo posible, lo cual permite tener un sistema abierto. Lo anterior significa que el servidor será portable bajo diferentes plataformas de hardware y software, y será un ejemplo de la tecnología que puede ser usada a en el resto de las funciones del sistema.

Los sistemas realizados en México para plantas generadoras de energía no utilizan las herramientas mencionadas anteriormente, y los sistemas comerciales no siempre permiten agregar software o hardware a sus sistemas.

II SISTEMAS DISTRIBUIDOS EN TIEMPO REAL

Los Sistemas Distribuidos En Tiempo Real se encuentran en una variedad de aplicaciones industriales, militares, bancarias, y de otros servicios como reservaciones en líneas aéreas etc. Nos interesa en el presente trabajo las aplicaciones industriales y en particular los sistemas de supervisión y control de centrales de energía, a los cuales hemos llamado SADREs. Estos sistemas son también denominados usualmente Sistemas de Control Distribuido en Tiempo Real, cuando está incorporada la función de Control al sistema, y se dice que un sistema de control distribuido tiene tres [Kagan 92] facetas para su estudio; el sistema de control y supervisión, el sistema de cómputo y el sistema de manejo de información. La característica principal de estos sistemas es que deben de registrar los eventos que suceden en el medio y controlar el proceso por medio de algoritmos en tiempo real. Debido a la naturaleza de las plantas, se necesita que la supervisión y el control se localice en diferentes áreas y sea operado por muchos individuos, lo anterior nos lleva a pensar que estos sistemas deben ser distribuidos. Se puede decir que la siguiente generación de sistemas de control en tiempo real [Arvind 91] cumplirán lo siguiente; serán sistemas distribuidos basados en redes locales, tendrán restricciones temporales, serán de naturaleza dinámica, deberán ser predecibles y la intercomunicación entre tareas tendrá implícitas restricciones temporales.

Los primeros sistemas distribuidos fueron estructurados desde el punto de vista que la interfase con la red era como otro dispositivo de Entrada-Salida (I/O). Esto es lo que se llama el modelo `file model' y este modelo persiste en el modelo de OSI. En la actualidad se usan interfases `procedure like' para sistemas distribuidos que en su forma mas sencilla consiste de un mensaje de requerimiento de un proceso cliente hacia un proceso servidor, seguido por un mensaje respuesta en la dirección opuesta. El anterior se denomina el modelo cliente/servidor [Mullender 89] o el modelo de operación remota. Cuando los requerimientos y las respuestas son estructuradas con una sintaxis de procedimiento, de tal manera que el cliente llama a una rutina intermedia o `stub' que coloca los argumentos de entrada en el mensaje de requerimiento, lo envía, espera por la respuesta, regresa los argumentos de salida y el resultado de la función, hablamos del modelo de llamadas a procedimiento remoto (RPC). Este último esquema tiene ventajas además de transportabilidad y estandarización, debido a que existen normas de facto (ONC y NCS) para sistemas distribuidos que lo soportan.

Todos los Sistemas Operativos en Tiempo Real modernos proporcionan facilidades de `multitasking', estas difieren en el tipo de modelo de memoria utilizado. Algunos proporcionan un modelo de memoria global, donde todos los procesos tienen acceso inmediato a todo el código y datos. En este modelo, el compartir el código y los datos es automático, se simplifica la programación y se reducen los tiempos de conmutación entre tareas, y usualmente en tales sistemas las áreas de memoria de las tareas no están protegidas unas de otras. Otros Sistemas Operativos proporcionan un modelo de memoria protegida que define para cada proceso su propio código y datos que son protegidos de otros procesos. En este modelo, el `kernel' también es protegido de los procesos de usuario, y las llamadas al sistema se deben hacer sólo por caminos controlados. Los sistemas modernos proporcionan lo mejor de los dos usando los llamados Procesos de Peso Ligero (LWPs). Un sistema con LWPs es principalmente un sistema con memoria protegida donde cada proceso tiene su memoria privada y protegida, pero cada proceso puede contener varios hilos de activación o LWPs con su propio `program counter' y pila, y por lo tanto su propio flujo de ejecución. En contraste los HWPs o Procesos de Pesados, no comparten memoria entre ellos, tienen un solo hilo de activación o flujo de ejecución, y tienen un solo PCB (Process Control Block) donde se guardan los valores de los registros de propósito especial, los de propósito general, los archivo abiertos, los candados en uso y otros recursos que el proceso está usando. La ejecución de varios hilos de activación en un mismo proceso tiene la ventaja de acceso inmediato a áreas de memoria común dentro del proceso. Esto elimina la necesidad de mecanismos especiales para acceso a memoria compartida, que es la desventaja del esquema del modelo de memoria protegida. De la misma manera se puede asegurar de varias formas la consistencia de la memoria compartida por medio de mecanismos de exclusión mutua. Este tipo de sistemas tiene beneficios substanciales para desarrollos en tiempo real, debido cada LWP puede tener su propia e independiente prioridad y se puede bloquear sin afectar que continúe la ejecución de los demás LWPs. Algo que también es importante, para los sistemas en tiempo real, en los LWPs es que el tiempo de creación, bloqueo, reactivación y destrucción de un LWP es bastante barato debido a que estos no cargan con toda la burocracia de un proceso pesado, puesto que comparten los recursos del sistema con el HWP al que pertenecen.

Sistemas Distribuidos

Se puede definir un Sistema Distribuido es una colección de varias computadoras autónomas conectadas por un subsistema de comunicación e integradas lógicamente por un Sistema Operativo Distribuido. De tal manera que los recursos del sistema sean manejados en forma global, es decir, que los usuarios no se enteren (si no quieren) de donde se almacenan los archivos o por quien son ejecutados los programas, y tal integración se puede lograr en varios niveles, con lo que se estima el grado de distribución del sistema.

Debido a que normalmente las computadoras de un sistema como el anterior no cuentan con una memoria común, e inclusive puede tratarse de diferentes tamaños, hardware, Sistemas Operativos, y Sistemas de Comunicación, es de mucha importancia que el sistema de comunicación sea capaz de enviar y recibir mensajes que sean comprendidos por elementos tan diversos. En general un Sistema Distribuido permite que varios procesos repartidos en una colección de computadoras, se comuniquen para alcanzar un objetivo común.

La característica mas importante de los Sistemas Distribuidos es el sistema de comunicación entre procesos. El tipo de sistema de comunicación que nos interesa es el de redes de computadoras autónomas, y en particular Redes Locales (LANs). Para lograr interconectar por medio de redes sistemas con diferentes arquitecturas y de diferentes proveedores, se ha definido un modelo de referencia "Open Systems Interconnection" de la "International Standards Organization", conocido como modelo ISO/OSI, este consta de siete capas de protocolos, que son : Aplicación, Presentación, Sesión, Transporte, Red, Liga de Datos, y Físico. Dentro del modelo ISO/OSI, la capa que juega un papel crítico en la velocidad de comunicación y la garantía de recepción, es la de Transporte, puesto que dependiendo si se escoge aquí un protocolo orientado a conexión, como TCP, o no-orientado a conexión, como UDP, se establecen diferentes características de la comunicación. También es importante la topología de la red, que en una LAM principalmente son de anillo o de "bus", con diferentes mecanismos en cada una. Un primer esfuerzo de estandarización de redes locales fue la definición, por un grupo de fabricantes, de Ethernet, que define una topología de "bus", pero el IEEE definió a partir del modelo OSI/ISO el standard IEEE 802 LAN, que especifica las capas física y de liga de datos. En los sistemas distribuidos es muy importante el modelo de comunicación seleccionado, el modelo de referencia ISO/OSI presenta muchas interacciones entre las capas y se debe tomar como una guía, no como una obligación.

La investigación en Sistemas Distribuidos se ha centrado principalmente en resolver problemas de compartir recursos, aumentar nodos o servicios al sistema, y como sobrevivir a la perdida de componentes del sistema (fault tolerance)

Un aspecto importante son los esquemas de interacción entre procesos. Se puede hablar de los siguientes paradigmas :

La forma mas popular de repartir el control de un Sistema Distribuido es precisamente repartirlo entre los procesos del sistema. Así mismo existe la tendencia de que muchas de las funciones del Sistema Operativo se realizan dentro de los procesos, y de esta forma existen procesos especialistas en realizar ciertas funciones comunes y otros que necesitan de estos especialistas. Los procesos entonces se clasifican como procesos clientes y procesos servidores. Como funciona lo anterior es por medio de mensajes que se envían entre clientes y servidores, por ejemplo un cliente envía una petición a un servidor, quien realiza el trabajo requerido y regresa un mensaje de respuesta al proceso cliente. Lo anterior nos dice que si existe un servidor por recurso o grupo de recursos, los clientes tienen que competir por el uso de los servidores. El modelo anterior es llamado Cliente/Servidor, y se puede definir como aquel donde un Servidor es un subsistema que proporciona un tipo especial de servicio (normalmente el acceso a un recurso compartido) a procesos Clientes no identificados a priori. Este modelo presenta problemas de cuellos de botella en los servidores, centralización de recursos, y redundancia al tratar de resolver los primeros dos.

El modelo integrado, donde cada proceso del sistema contiene todas las funciones (que usualmente están en el Sistema Operativo), de tal manera que la configuración del sistema implica sólo el activar o desactivar funciones, y en una configuración específica puede parecer que se trata del modelo cliente/servidor original.

El Modelo "file model" o "pipe model" donde se transfieren datos entre los procesos en un esquema FIFO (First In First Out) y el manejo interno de los "pipes" es como si fueran archivos. La ventaja de los "pipes" es que permiten enviar cantidades grandes de datos, por medio de una llamada remota de entrada o salida, a nodos remotos sin bloquear al proceso llamador. Los "pipes" se pueden usar desde procesos emparentados (padre-hijo) sin necesidad de un identificador, y por medio de un identificador de "pipe" se puede accesar un "pipe" desde procesos no emparentados.

El modelo de Llamadas a Procedimientos Remotos (RPCs), donde se permite que los procesos llamen a un procedimiento en una computadora remota, y se utiliza la misma terminología cliente/servidor para designar al proceso llamador y al proceso llamado. Mas adelante se describe con mas detalle este modelo puesto es el que se ha seleccionado para implementar el

prototipo de servidor.

Con respecto a los modelos de implementación de sistemas distribuidos, se puede hablar de dos modelos : el modelo de procesos y el modelo de objetos.
El modelo de procesos está basado en dos entidades básicas, el proceso y el mensaje, los subsistemas de servicios se realizan con conjuntos de procesos para este fin, y en general los procesos interactúan entre si por medio de mensajes.

El modelo de objetos trata de caracterizar las entidades del sistema en la forma como el ser humano percibe el mundo. Aquí el elemento básico es el objeto, que consiste de estructuras de datos y operaciones definidas sobre esas estructuras, y debido a que el objeto es un tipo de datos abstracto se ocultan los detalles internos (mecanismos de sincronización, de comunicación, etc..) del objeto para el usuario. La principal desventaja de el modelo de objetos es el pobre desempeño en tiempo de ejecución.

Sistemas en Tiempo Real

Los Sistemas en Tiempo Real se definen como [Stankovic 88] aquellos en los que la validez del sistema no sólo depende de los resultados de los cálculos y de la lógica de los programas, sino también de el momento en que los resultados son producidos o la lógica es ejecutada. Existe una gran variedad de sistemas que encajan en esta definición, entre los que se encuentran sistemas de comunicación multimedia, sistemas de procesamiento de transacciones en línea, sistemas para control de manufactura y sistemas de control de procesos. En nuestro caso trataremos los sistemas ce control de procesos en tiempo real, que se puede describir abstractamente como un ciclo de retroalimentación que consiste de cuatro componentes : un proceso a controlar, un controlador, sensores y actuadores. Los sensores proporcionan a el controlador información acerca del estado actual del proceso a controlar. El controlador es un sistema de procesamiento de información que hace uso de la información proporcionada por los sensores para compensar la diferencia entre el estado actual y el estado deseado del proceso a controlar. Los actuadores es el medio por el que se realizan las acciones dictadas por el controlador.

En la mayoría de los sistemas de control de procesos en tiempo real, tales como sistemas de control de plantas de energía o sistemas de control de plantas químicas, el controlador incluye elementos humanos además de computadoras, y estos son llamados de ciclo abierto. Estos elementos humanos toman decisiones de acuerdo a los datos procesados de los sensores por computadores. Cada vez mas se reducen las intervenciones humanas, y esto lleva restricciones mas fuertes en cuanto a la dinámica del sistema y a las metas a cumplir.

Los sistemas de control de procesos en tiempo real en la actualidad se caracterizan como sigue :

a) Son Sistemas Distribuidos basados en Redes Locales (LAN) debido a las distancias entre los equipos son cortas, y por medio de la distribución de trabajo se pueden satisfacer requisitos de actuación, confiabilidad y funcionabilidad.

b) Son sistemas con restricciones temporales, donde las tareas pueden ser periódicas o tener asociado un período de caducidad (deadline).

c) Son cada vez con mas frecuencia de naturaleza dinámica, donde, además de las tareas que son activadas desde el arranque del sistema, se necesita manejar tareas que son activadas con la dinámica del ambiente o a petición de los usuarios.

d) Son sistemas predecibles, donde se puede determinar si los restricciones temporales de una tarea pueden cumplirse antes de permitir ejecutarla.

e) Necesitan mecanismos de intercomunicación entre tareas que deben cumplir también con restricciones temporales que se traducen en períodos de caducidad en los mensajes u otros mecanismos.

Llamadas a Procedimientos Remotos

Las llamadas a procedimientos remotos o RPCs, permiten que un cliente ejecute procedimientos en otra computadora de la red o servidor. Es decir que el modelo cliente/servidor se simplifica y aumenta sus potencialidades en contraste con los "sockets", que la forma antigua de realizar tal comunicación.

Como funciona un esquema cliente servidor por medio de RPCs es como sigue :

1) El cliente y el servidor corren en dos procesos separados, y no necesariamente corren en la misma máquina.

2) Ambos procesos se comunican por medio de "stubs", uno para el cliente y uno para el servidor. Estos "stubs" son código que contiene funciones para mapear las llamadas locales a llamadas remotas en la red.

La intercomunicación entre procesos es una parte fundamental en el modelo cliente/servidor. Las comunicaciones usando "sockets" de UNIX son además de difíciles de desarrollar, son difíciles de mantener y desplegar en aplicaciones de múltiples procesos, especialmente cuando la aplicación está distribuida a través de la red, y cuando al final se logra establecer la comunicación surjan problemas de manejo de errores, portabilidad, etc. y a pesar de que inclusive muchos sistemas no UNIX incluyen librerías para lo anterior, estas no son facies de usar.

La opción está entre batallar con la representación de datos, el protocolo para envió de mensajes, etc. o usar RPCs que ocultan todos los detalles anteriores a los clientes y a los servidores.

Lo que sucede en la comunicación vía "sockets" es que los servicios se reportan en un servicio de nombres o "deamon", y este le da a cada cliente la dirección donde pueden abrir un canal de comunicación hacia el servidor. Después el servidor puede aceptar o rechazar la petición de servicio, enviado respuestas.

A diferencia de los "sockets" en una comunicación con RPC independiente del transporte o TIRPC, el TIRPC lleva control de las direcciones a nivel simbólico (nombres de nodos, transportes, etc..) y en los "sockets" los servicios se cargan cuando se inicializa el sistema con los valores del archivo etc.

Los siguientes son los pasos que se necesitan para que un cliente usando RPCs en Open Network Computing (ONC) de Sun pueda llamar a un servidor :

1. El servidor debe registrar la dirección donde el espera las peticiones de servicios en el protmapper (que es el proceso

responsable de mapear los servicios a puertos) así como los numeros de programas y versiones que el servidor pretende atender.

2. El cliente pregunta al portmapper (tiene el puerto 111) cual es el puerto del servidor que necesita (handle de cliente).

3. El cliente puede con el puerto del servidor hacer un llamado a procedimiento remoto y el servidor regresa una respuesta.

El modelo de referencia OSI/ISO , como ya dijimos anteriormente, define siete capas de comunicación, y los componentes de RPCs en ONC pueden describirse de la siguiente manera :

1) La aplicación se coloca en la capa superior (capa de aplicación)

2) El esquema de representación de datos XDR se sitúa en la capa de presentación.

3) La librería de servicios de RPCs en la capa de sesión.

4) Los protocolos TCP y UDP en la siguiente, la de transporte.

5) El protocolo IP en la capa de red.

6) Por último la interfase de hardware en el nivel físico y de liga de datos.

Se puede decir que hay dos tipos de servidores, los servidores con_estado y los servidores sin_estado. Los servidores sin_estado no mantienen información o estado de las interacciones que ha tenido con los clientes y por el contrario los con_estado si lo hacen. Por ejemplo si el resultado de un servicio depende de servicios anteriores del mismo cliente se dice que el servidor es con_estado y debe de recordar los pasos que siguió. Si un servidor sin_estado aborta su recuperación es inmediata, sin embargo para un servidor con_estado deberá hacer lo necesario para colocarse en el estado en que estaba antes de la caída.

La ventaja principal del manejo de clientes y servidores por medio de RPCs es que la intercomunicación entre procesos está oculta para el programador, y mas cuando ni siquiera tiene que hacer el código de los "stubs". Para lo anterior existen compiladores que en base a una descripción de protocolo, producen los "stubs" de cliente y servidor.

Procesos de Peso Ligero

Existen en general tres tipos de procesos que pueden ser soportados por un sistema distribuido : de peso ligero o LWPs, de peso pesado o HWPs, y de peso medio o MWPs. Esta división está basada en la forma como comparten memoria.

Por ejemplo :

Un proceso ligero puede tener dos flujos de ejecución (o hilos de activación) que comparten datos de memoria común y datos de una pila común.

Dos proceso pesados tienen cada uno dos hilos de activación privados cada uno sus propios datos de memoria privada y cada uno su propia pila privada.

Un proceso de peso medio puede tener dos hilos de activación, que comparten datos de memoria común y que tienen cada hilo su propia pila privada.

Se puede decir entonces que un LWP es aquel que contiene mas de un hilo de activación y todos ellos usan el mismo código, datos y pila. Los hilos de un LWP se ejecutan en el mismo espacio de direccionamiento esto significa que el hardware de direccionamiento permite que los hilos accesen las mismas localidades de memoria.

El nombre de ligero, viene de que la creación, existencia y destrucción del hilo (proceso), así como las primitivas de sincronización son bastante baratas en tiempo, en comparación con las primitivas de procesos pesados.

Si los LWPs se utilizan en lugar de los HWPs para problemas donde sea necesaria concurrencia, la utilización del CPU se mejora notablemente. Los principales motivos por los que se desea tener memoria compartida entre procesos de usuario son :

1) Se reduce la burocracia de intercomunicación entre procesos y transiciones de proceso a proceso (cambio de contexto).

2) Se pueden desarrollar servidores que manejen peticiones de clientes en paralelo en lugar de serializarlas o crear un proceso servidor por cliente.

3) Se pueden manejar dispositivos lentos cono discos, cintas, terminales, impresoras, etc. de tal manera que el programa trate de realizar otra tarea mientras los datos están listos.

4) Se pueden programar acciones concurrentes que el usuario espera no se hagan en serie.

5) Muchas aplicaciones necesitan que varios procesos compartan un área común de memoria.

Los procesos de peso ligero implementados en Sun OS permiten lo siguiente :

1) La asignación de prioridades a cada hilo.

2) Se crea una cola de despacho de procesos por cada prioridad con política FIFO.

3) Se puede cambiar la prioridad de un proceso a tiempo de ejecución.

4) Se puede reorganizar una cola de despacho.

5) Se cuenta con mecanismos de sincronización como semáforos, monitores y mensajes.

6) Cada proceso se crea a partir de una función y por lo tanto tiene su memoria de datos privada y su pila privada, además de poder accesar a la memoria común y a una pila común.

Resumen

Los Sistemas de Supervisión y Control se caracterizan por ser sistemas distribuidos que utilizan redes locales, debido a la localización geográfica del sistema. Son también sistemas en tiempo real debido a las exigencias de respuesta al proceso.
Un aspecto importante a considerar en los sistemas distribuidos es la comunicación entre los nodos, y una forma de estructurar tal comunicación que presenta ventajas de abstracción y normatividad es el paradigma de Llamadas a Procedimientos Remotos o RPCs. Dentro de un sistema en tiempo real es conveniente contar con mecanismos de memoria compartida y facilidades de `scheduling' que cumplan con las restricciones de tiempo, y lo anterior se puede lograr con programación de alto nivel usando el paradigma de Procesos de Peso Ligero o LWPs.

III NORMATIVIDAD

Los estándares en la industria informática pueden ser fundamentalmente de dos tipos : estándares "*de facto*" y estándares "*de jure*" [Corbin 91].

Los estándares "*de facto*" son aquellos que se establecen por la preferencia de los usuarios, los estándares "*de jure*" son aquellos generados por organismos que se dedican a establecer estándares. Tales organismos normalmente están apoyados por agencias de gobierno, compañías, o grupos de usuarios.

La normatividad en los sistemas SADRE tiene especial importancia para desarrollar productos de software que puedan ser trasladados y competir en el mercado. En este caso se puede enfocar la normatividad desde el lenguaje de programación y las rutinas de acceso a los servicios del Sistema Operativo, tanto para los servicios comunes, como a los específicos de la aplicación. En estos servicios específicos podemos contar las facilidades para Programación En Tiempo Real, el acceso a Procesos de Peso Ligero, y acceso a Llamadas a Procedimientos Remotos.

En el caso del acceso a los servicios del Sistema Operativo existe el estandard "*de jure*" POSIX.1, donde se define una interfase estandard con el sistema operativo.

Dentro de la familia de estándares POSIX, en el bosquejo de los estándares para extensiones en tiempo real a POSIX.1, llamado POSIX.4 o *IEEE 1003.4*, se proporcionan facilidades para compartir memoria, garantizar exclusión mutua, scheduling, manejo de relojes y cronómetros entre otras cosas. El estandard POSIX.4a o *IEEE 1003.4a* define las extensiones para soportar Procesos de Peso Ligero y el bosquejo actual incluye scheduling de LWPs, primitivas de sincronización, cancelación de LWPs, y área de datos dedicadas a los LWPs.

Debido a que las implantaciones de los mecanismos de RPCs con una interfase de aplicación para programación (API) están normalmente embebidos en un ambiente de computo distribuido, se mencionarán las dos mas importantes implantaciones comerciales de ambientes distribuidos (que por su aceptación se consideran normas "*de facto*"), que son: El Open Network Computing de Sun Microsystems y el Network Computer System de HP y Apollo. Ambos tienen grupos importantes de proveedores que lo soportan, siendo actualmente el mas usado el ONC de Sun.

Norma POSIX.1 de Interfase con Sistemas Operativos

En 1968 los laboratorios Bell de AT&T empezaron a trabajar en el sistema operativo UNIX. Esto permitió que un sólo sistema operativo corriera en plataformas de hardware de diferentes proveedores, sin embargo UNIX se ha desarrollado en diferentes sentidos : el sistema V de AT&T, el sistema de Berkeley Software Distributions, Xenix, etc.. Ninguno de los estilos anteriores son iguales y el comportamiento de cada estilo no está bien definido.

Actualmente existe una disputa entre los sistemas operativos para ganar el mercado, y los contendientes son Unix Systems Laboratories System V, Open Software Foundation OSF/1, DEC VAX/VMS y Microsoft OS/2. Ahora mismo todos ellos han acordado seguir los estándares POSIX.

POSIX está basado en el UNIX System V y en Berkeley UNIX, pero no es en si mismo un sistema operativo, sino que POSIX describe la interacción entre la aplicación y el sistema operativo. POSIX no establece como escribir programas de aplicación o como diseñar sistemas operativos, en vez de eso POSIX define la interfase entre las aplicaciones y las bibliotecas del sistema. POSIX es un estandard independiente de un proveedor, una arquitectura o un sistema operativo.

El nombre formal del estandard es : IEEE 1003.1-1988 Portable Operating System Interface for Computer Environments. Se le conoce comúnmente como POSIX por hacer las siglas similares a UNIX. En 1990 POSIX se convirtió en un estandard internacional ISO/IEC 9945-1:1990 y el Gobierno de los Estados Unidos ha adoptado tal estandard como el estandard federal para proceso de información FIPS 151, y se espera que organismos europeos lo hagan también suyo. Todo lo anterior ha llevado a que diferentes proveedores de equipo y software anuncien su apoyo a POSIX.

En realidad POSIX es una familia de estándares de los cuales nos interesan los relacionados con sistemas distribuidos y con sistemas en tiempo real, los siguientes son los que se pueden considerar para realizar una aplicación en de sistemas distribuidos en tiempo real portable :

POSIX.1 Define la interfase entre los programas de una aplicación portable y el sistema operativo, basado en los modelos históricos de UNIX. Esta interfase consististe de una librería de funciones que normalmente son implantadas por medio de llamadas al sistema.

La primera versión de este estandard está escrito en términos del lenguaje C.

POSIX.4

Es un conjunto de extensiones para tiempo-real al estandard POSIX.1 y está en desarrollo. Los últimos bosquejos incluyen lo siguiente :

- Semáforos binarios
- Candados para memoria de procesos
- Archivos mapeados a memoria y memoria compartida
- Despacho de procesos por prioridades
- Extensiones de señales en tiempo real
- Relojes y Despertadores
- Envío de mensajes entre procesos
- Entrada/salida sincronizada y asincrona
- Archivos de tiempo real

Muchas de estas funciones son también útiles en aplicaciones que no son de tiempo real y a pesar de que todavía no es un estandard, muchos proveedores anuncian sus productos como compatibles con los bosquejos.

POSIX.4a

Este estandard agrega funciones que permiten el manejo de hilos de activación (o procesos de peso ligero) y el último bosquejo incluye las siguientes funciones :

- Despacho de hilos de activación
- Primitivas de sincronización entre hilos de activación
- Destrucción de hilos de activación
- Memoria local a hilos de activación

Normas de Facto ONC y NCS para RPCs

Los sistemas de RPCs están siempre embebidos en un ambiente de computación distribuida, siendo las dos mas importantes implantaciones comerciales de ambientes distribuidos el Open Network Computing (ONC) de Sun y el Network Computing Environment de Apollo y HP.

Cuando la "Open Software Foundation" hizo su llamado de "Request for Technology" para las herramientas de su ambiente DECORUM DCE, dos grupos sugirieron mejoras a sus respectivos ambientes de computación distribuida. Por un lado estaba ONC con AT&T, Sun, Novell, y Netwise apoyando nuevos desarrollos en el sistema de RPCs de ONC. Por otro lado estaba las mejoras a NCS, que es el ambiente de HP y Apollo, apoyados por IBM, DEC y Microsoft.

Parecía que la OSF decidiría por la propuesta de ONC porque este ambiente ya estaba establecido en el mercado e inclusive los códigos fuente estaban disponibles al público sin costo. Finalmente la OSF seleccionó pedazos de ambas propuestas para incluirlas en su DCE. Sin embargo seleccionó a NCS como el mecanismo de principal de RPCs.

A pesar de lo anterior, mucho de lo propuesto por NCS está orientado a desarrollos futuros y el hecho es que actualmente no se puede conseguir el ambiente NCS completo.

Considerando lo anterior podemos decir que el standard "*de facto*" por el momento es el ONC de Sun OS. La siguiente tabla apoya la aseveración anterior, mostrando las características mas importantes de ambientes distribuidos y como las manejan NCS y ONC.

FUNCIÓN	ONC (SVR4 de AT&T)	NCS (rev. 3.0)
Distribución	**Sin costo**, fuentes incluidos	Código binario, propietario opcional con Sistema Oper.
Sistemas Operativos	UNIX-SVR4,Sun OS,HP-UX,Domain/OS,ULTRIX, AIX,DOS,OS/2,MVS,VM	UNIX-Domain/OS, HP-UX, VMS, ULTRIX
Transportes	TLI,TCP,UDP,PC LAN Netwise & Novell, 3Com, Banyan, PC-NFS	TCP,DDS,PC LAN Microsoft
Representación de datos	XDR (Normalizado) un nivel canónico	NDR (No-Normalizado) 16 niveles canónicos
Apuntadores	Si	Sólo pasado por referencia
Tipos de datos	Los de C, estructuras arbitrarias y datos opacos	Los de C, estructuras de datos elementales
Instalaciones	1.2 M	200,000
Licencias	290	200
Implementaciones	90	10
Protocolo de RPC	Estado en el transporte, a-conexión o no-a-conexión, tiempos de vencimiento (time outs), control de reintentos	Servidores Con-estado, a-conexión o no-a-conexión, no hay tiempos de vencimiento, no hay reintentos
Tipo de relación cliente/servidor	Persistente (TCP, UDP), no-persistente (UDP)	Completamente asociado, no-asociado
Comunicación por difusión (broadcast)	Si	Si
Procesamiento múltiple de peticiones al servidor	Peticiones en cola, operación manual de multitareas por medio de procesos pesados	Igual, mas LWPs cuando está disponible CPS (Concurrent Programming Support)

Resumen

La norma POSIX.1 define la interfase entre un programa de aplicación portable y el sistema operativo, esta norma se basa en los modelos tradicionales del sistema UNIX y consiste de una biblioteca de funciones que frecuentemente se implantan como llamados al sistema. Las normas para tiempo real de POSIX son la 1003.4 y 1003.4a, tratan las extensiones para relojes, cronómetros, scheduling, exclusión mutua y manejo de Procesos de Peso Ligero. Los sistemas ONC y NCS permiten el paradigma de Llamadas a Procedimientos Remotos y se puede decir que el estandard "*de facto*" es el ONC por estar disponible mayor cantidad de sistemas operativos y por ser de distribución gratuita.

IV APLICACIÓN

Un Sistema de Adquisición de Datos y Registro de Eventos (SADRE), es un sistema de supervisión que informa a los operadores de una central de energía el comportamiento del proceso.

Dentro de las funciones de un SADRE está la de Registro Histórico, la cual se encarga de registrar los valores de las señales adquiridas por el SADRE de acuerdo con un plan de muestreo preestablecido y proporcionar resúmenes o extracciones de los datos registrados, así como otro tipo de informaciones relacionadas.

Se seleccionó la función de registro histórico para realizar un servidor distribuido porque es una función típica de un SADRE y porque la especificación de tal función permite tiempos de respuesta no muy estrictos, pero que requiere ser atendida en seguida de la función de adquisición, que es la de mas alta prioridad.

El servidor consta en realidad de dos servidores, uno para las funciones propiamente de registro y otro para las funciones de manejo de archivos, se realizó bajo el sistema operativo Sun OS, utilizando las librerías de llamadas a procedimientos remotos y de procesos de peso ligero.

Los resultados obtenidos muestran que sí se pueden cumplir las metas de tiempo de respuesta tanto en el registro como en el manejo de archivos.

Sistemas de Adquisición de Datos y Registro de Eventos

Un Sistema de Adquisición de Datos y Registro de Eventos ofrece a los operadores de una central generadora de energía una herramienta de supervisión y documentación del comportamiento del proceso de generación de energía. La confiabilidad y disponibilidad de un SADRE debe ser mayor del 98 % , lo que significa que debe estar en operación 357.7 días al año.

El sistema mantiene informado al operador del estado histórico y actual que guarda el proceso. un SADRE no toma acciones de control, sino que sólo presenta y registra información en forma oportuna, para que después de ser analizada por los operadores, éstos tomen las acciones que consideren pertinentes.

En los SADRES con arquitecturas repartidas, se eliminan los procesadores centrales y sólo se cuenta con unidades de adquisición y puestos de mando, que se interconectan por medios rápidos. Los puestos de mando se desarrollan con estaciones de trabajo que tienen capacidad de procesamiento, almacenamiento, y de presentación de información. Dependiendo de cada puesto de mando se tienen funciones de registro, informes periódicos, guías de operación, estado de la central, diagnóstico de alarmas, evaluación del estado operacional del equipo, análisis estadístico y en frecuencia de los datos recibidos, predicción de estados e informe global del funcionamiento de la central.

La función que se ha utilizado para realizar un prototipo de servidor distribuido, es la de registro histórico, la cual se detalla en el siguiente punto.

Función de Registro Histórico de un SADRE

La función de Registro Histórico tiene, los siguientes requisitos funcionales :

A) Se requiere llevar el registro de cualquier variable, donde se consideran dos etapas, una de definición y otra de colección. En la primera etapa se definirán las variables a las que se dará el seguimiento histórico, el período de muestreo y el tiempo que durará la colección de los valores, así como los estados operativos. La información será almacenada en disco y podrá ser recuperada para presentarse en pantalla en la segunda etapa.

B) Con respecto al registro de datos en disco, se requiere que el usuario seleccione el modo de almacenamiento y la recuperación de datos en los archivos históricos; el almacenamiento se suministrara para cualquier tipo de dato y se tendrá con el suficiente detalle para reconstruir su información, se debe permitir lo siguiente:

1. Almacenar cualquier señal.

2. Suministrar información para determinar si se han almacenado valores inválidos o datos erróneos de cualquier variable.

3. Capacidad para transportar los datos a un medio externo de almacenamiento. (cinta magnética o cartucho)

C) Para la recuperación de los datos se requiere que esta función actúe como interfase entre los datos almacenados de una variable y el proceso que solicita esos datos, para lo cual la función debe cumplir con lo siguiente:

1. Capacidad para manejar solicitudes de valores de varias variables para un tiempo de cómputo dado.

2. Recuperar datos en tiempo real (recién almacenados), datos de historia reciente y archivar datos.

Los siguientes requisitos de software están basados en los requisitos establecidos para el Sistema de Control Distribuido (SICODI) de la central de ciclo combinado de Gómez Palacio Dgo., dado que la Función de Registro Histórico (FRH) no está involucrada en acciones de control, se puede decir que la función forma parte del SADRE del SICODI.

El término control 'distribuido' se refiere a que la lógica de control reside en los elementos finales (unidades de adquisición) y no en los nodos donde se realiza la supervisión, registro y despliegue. El SICODI consta de Estaciones de Trabajo conectadas por medio de una red Ethernet, dos de ellas están conectadas a concentradores que recogen las señales de las Unidades de Adquisición (UAs), una llamada estación principal y la otra de respaldo, el resto son llamadas estaciones de operador como se aprecia en la figura 1.

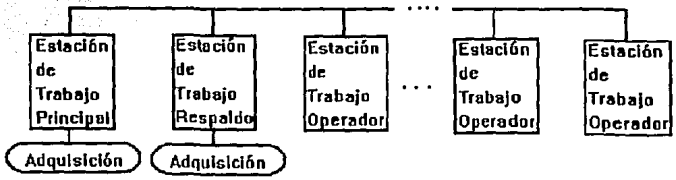

Figura 1

En la estación principal y en la de respaldo residen las funciones de Adquisición, Configuración de Planes, y Administrador de Base de Datos. Se pretende que el servidor de la FRH también se localice en estos nodos, y que de esta manera el resto de las estaciones de trabajo puedan accesar los servicios de esta función.

La FRH tiene relación con el Administrador de la Base de Datos, la función de Adquisición y la función de Configuración de Planes. La figura 2 muestra las interacciones de la FRH con otras funciones del SICODI.

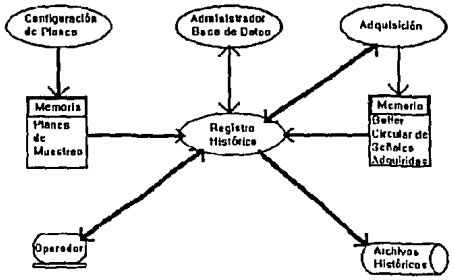

Figura 2

La Función de Adquisición de Datos (FAD) deposita cada segundo las señales que se adquirieron en un buffer circular. Este buffer tiene capacidad de contener 20 segundos de adquisición. El formato para la instancia de cada señal es el siguiente :

dirección	valor en cuentas	estampa	status
16 bits	12 bits	12 bits	8 bits

Donde la dirección es un número único que identifica la variable, el valor en cuentas es la cantidad registrada por el sensor, la estampa de tiempo es el momento en que se realizó tal adquisición y tendrá resolución de dos milisegundos, y el status indica la calidad de los datos o límites de operación. Cada segundo en el buffer consistirá de un encabezado que contiene la hora (día-mes-año hora:minuto:segundo), y 5,000 espacios para instancias de señal.

hora	$señal_1$	$señal_2$	. . . . .	$señal_{5000}$
8 bytes	6 bytes	6 bytes		6 bytes

El número máximo de señales que se adquieren por segundo es 5,000, por lo tanto cada segundo del buffer circular será de 30,008 bytes, y el buffer completo será de 600,160 bytes.

La FRH necesita saber si las variables (señales) definidas en el plan de muestreo, están también definidas en la base de datos.

La Función de Configuración de Planes (FCP) genera los planes de muestreo, tanto en disco como en memoria (Buffer) para que los accese la FRH. La interfase con la FCP es el área de memoria donde se encuentran los planes de muestreo. Cada plan de muestreo puede tener especificadas hasta 5,000 señales, y tiene el siguiente formato :

dir. $señal_1$	per. $señal_1$	. . .	dir. $señal_{5000}$	per. $señal_{5000}$
2 bytes	2 bytes		2 bytes	2 bytes

Donde el período está dado en milisegundos, y cuando la dirección de la señal vale cero significa que no hay mas señales en el plan. Se podrán tener en memoria (accesibles para la FRH) hasta 10 planes de muestreo.

Por lo tanto, cada plan se almacenará en 20,000 bytes y el total de el área de memoria reservada para los Planes de Muestreo será de 200,000 bytes.

El registro histórico debe tener la segunda prioridad mas alta después de la función de adquisición. Con esta condición se cuida que el registro histórico realice su actividad sin que pierda información, una vez que la función de adquisición termina de depositar los datos en memoria

La función de registro histórico lee y verifica la existencia en la base de datos de las variables que están definidas en el plan de muestreo, si alguna variable no existe, entonces la reporta inexistente con un mensaje, para las variables existentes aplica el registro según la frecuencia de muestreo.

la primera vez se registran todas las variables y después, sólo aquellas variables que cambian, si no hay cambio luego de transcurridos cinco segundos, entonces se registran todas las variables que no han cambiado, de esta manera se verifica que la variable se ha mantenido y que la función de registro histórico está operando normalmente.

El almacenamiento se hará en archivos diferentes de acuerdo al tipo de sesiones, que pueden ser arranque, operación normal y paro de la adquisición, esto con el fin de respaldar en forma controlada y separada la información de acuerdo con su interés.

Para llevar a cabo el análisis de los datos históricos, se contará con la capacidad de seleccionar los archivos históricos ya creados y con el fin de optimizar el almacenamiento permanente en disco, se podrán crear archivos menores (o de capacidad limitada) a partir de archivos que contienen datos almacenados masivamente en disco; En estos archivos de capacidad limitada, se almacena información que corresponda a :

a. Un rango de tiempo.

b. Una selección de variables.

c. Una combinación de las anteriores.

Como complemento, para el manejo de información histórica separada, se requiere permitir al operador manipular archivos históricos y crear archivos con de su interés.

En el diálogo del registro histórico, el operador podrá realizar las siguientes actividades:

a. Arrancar y parar la adquisición en caso necesario para aplicar un plan de muestreo.

b. Seleccionar planes de muestreo.

c. Consultar el espacio disponible en disco.

d. Copiar de un archivo histórico a otro un segmento (o rango) seleccionado, esto con la finalidad de analizar y guardar sólo la parte de interés.

e. Respaldar y bajar archivos de cinta.

f. Borrar archivos históricos de disco.

g. Seleccionar un archivo histórico para análisis.

h. Seleccionar un archivo histórico para análisis.

i. Mostrar archivos existentes en disco y cinta.

Diseño de un Servidor Distribuido para la Función de Registro Histórico

De acuerdo con lo mencionado en el punto anterior, la función de registro histórico se comunicará con las de adquisición de datos, la de manejo de base de datos y la de configuración de planes de muestreo. Para lo anterior se propone que cada una de estas funciones se localice en un proceso pesado y el servidor de registro histórico (que en realidad son dos servidores) se comunique con ellas por medio de llamadas a procedimientos remotos o por memoria o archivos compartidos. Una posible configuración, que cumple con los requisitos del punto anterior, es la mostrada en la figura 3.

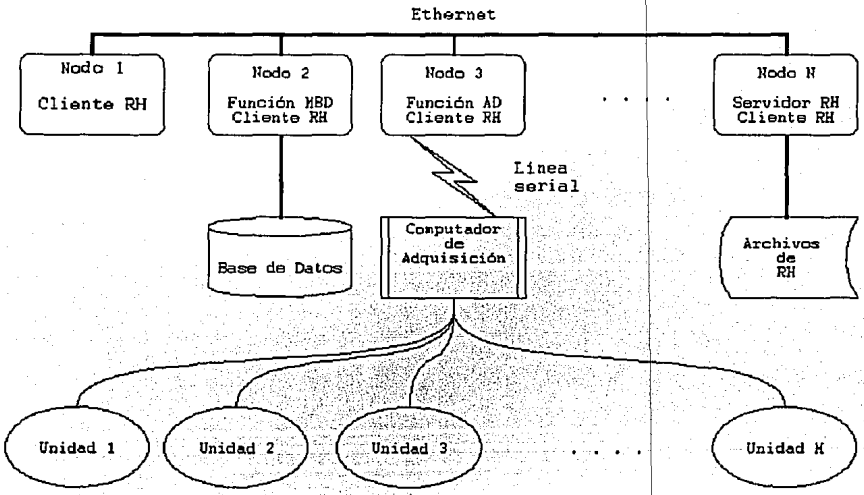

Figura 3

El servidor de registro histórico se encuentra sólo en un nodo, que es donde se encuentran los archivos de registro. En todos los nodos pueden existir procesos clientes, inclusive en el mismo donde está el servidor. Debido a que las funciones de adquisición, y manejo de base de datos se comunican con el servidor de registro histórico por medio de RPCs, éstas pueden localizarse en otros nodos, y el sistema puede ser fácilmente reconfigurable.

Lo anterior no limita a que se puedan tener varios servidores de registro histórico en diferentes nodos, con lo que se puede establecer una robustez del sistema hasta de un 100%, lo que significa tener replicado el sistema en cada nodo.

El servidor de registro histórico se diseñó en realidad en dos servidores :

a) El servidor de registro propiamente dicho, llamado RH. Que básicamente se encarga de cambiar planes de muestreo, detener o arrancar el registro, tomar los datos de la adquisición, y registrarlos en el archivo de *rastreo* de acuerdo con el plan de muestreo.

b) El servidor de manejo de archivos, llamado MA. Que se encarga de los vaciados y extracciones sobre el archivo de rastreo y de otros servicios relacionados con el manejo de archivos.

La figura 4 muestra como están relacionados RH y MA con los clientes y con los procesos externos. Ambos servidores crean procesos ligeros para atender a los clientes, un proceso por servicio, además de que el servidor RH crea al proceso colecta desde el principio de su ejecución.

En el caso de RH el hilo principal de ejecución siempre tiene prioridad MAXPRIO, la máxima prioridad dentro del proceso pesado, mientras que *colecta* tiene prioridad MAXPRIO-1, puesto que uno de los servicios que atiende el proceso pesado es precisamente el envío de muestras desde la adquisición. Todos los servicios inicialmente tienen MAXPRIO pero aquellos que no son críticos para el registro se rebajan a si mismos la prioridad a MINPRIO, como por ejemplo el servicio de listar los planes de muestreo.

Son de particular importancia el servicio *envia_muestras* y el proceso *colecta* , puesto que estos se comunican por medio de una lista circular y están sincronizados por medio de un monitor y dos variables de condición, que permiten el acceso a la memoria común (la lista circular) sin contradicciones y en secuencia.

En el servidor MA se localizan todos los servicios que tienen que ver con el manejo de archivos y los mas importantes de estos (de hecho son los que realmente consumen tiempo y pueden causar problemas) como son las extracciones basadas en rangos de tiempo, las extracciones basadas en variables, las extracciones mixtas, y los vaciados completos.

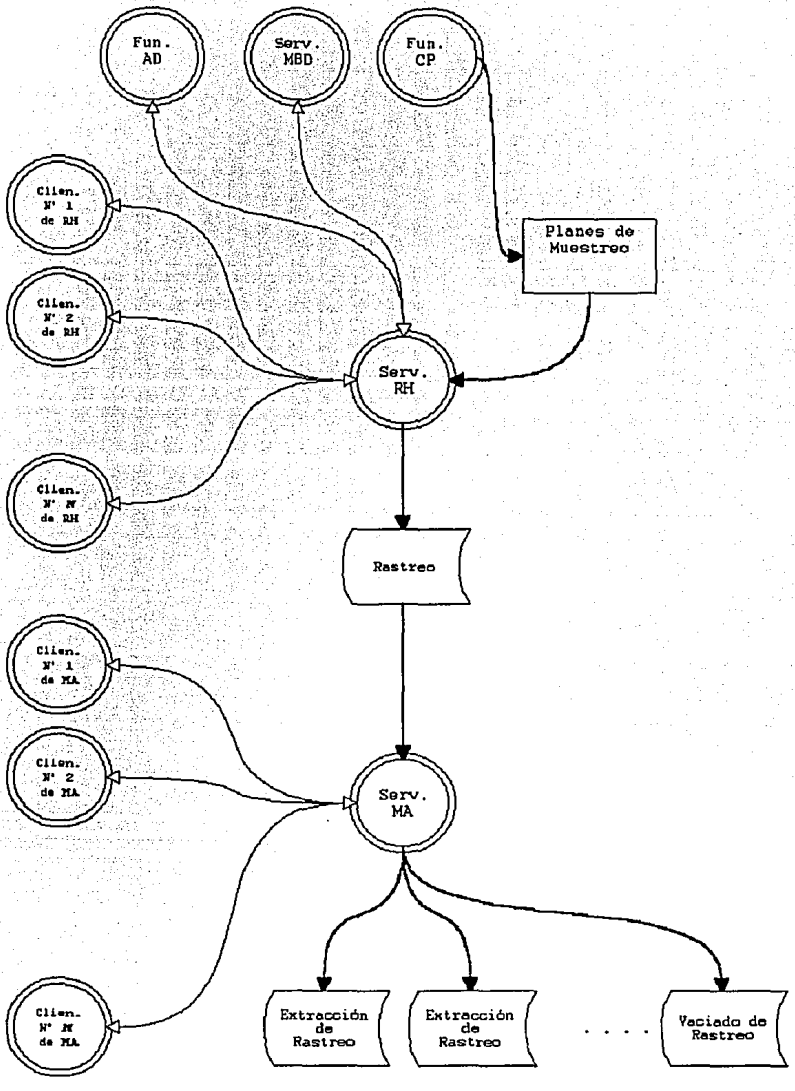
Fun. AD
Serv. MBD
Fun. CP
Clien. N° 1 de RH
Clien. N° 2 de RH
Clien. N° N de RH
Planes de Muestreo
Serv. RH
Rastreo
Clien. N° 1 de MA
Clien. N° 2 de MA
Clien. N° N de MA
Serv. MA
Extracción de Rastreo
Extracción de Rastreo
. . . .
Vaciado de Rastreo

Figura 4

Estos servicios se atienden por LWP que regresan inmediatamente al cliente un estatus de que la copia está en proceso pero no esperan a terminar la copia para decirle al cliente que tuvo éxito. Lo anterior se estableció debido a que las copias pueden tardar minutos cuando se trata de varios clientes a la vez que necesitan archivos de varios megabytes.

La figura 5 muestra como están relacionados internamente los servidores RH y MA :

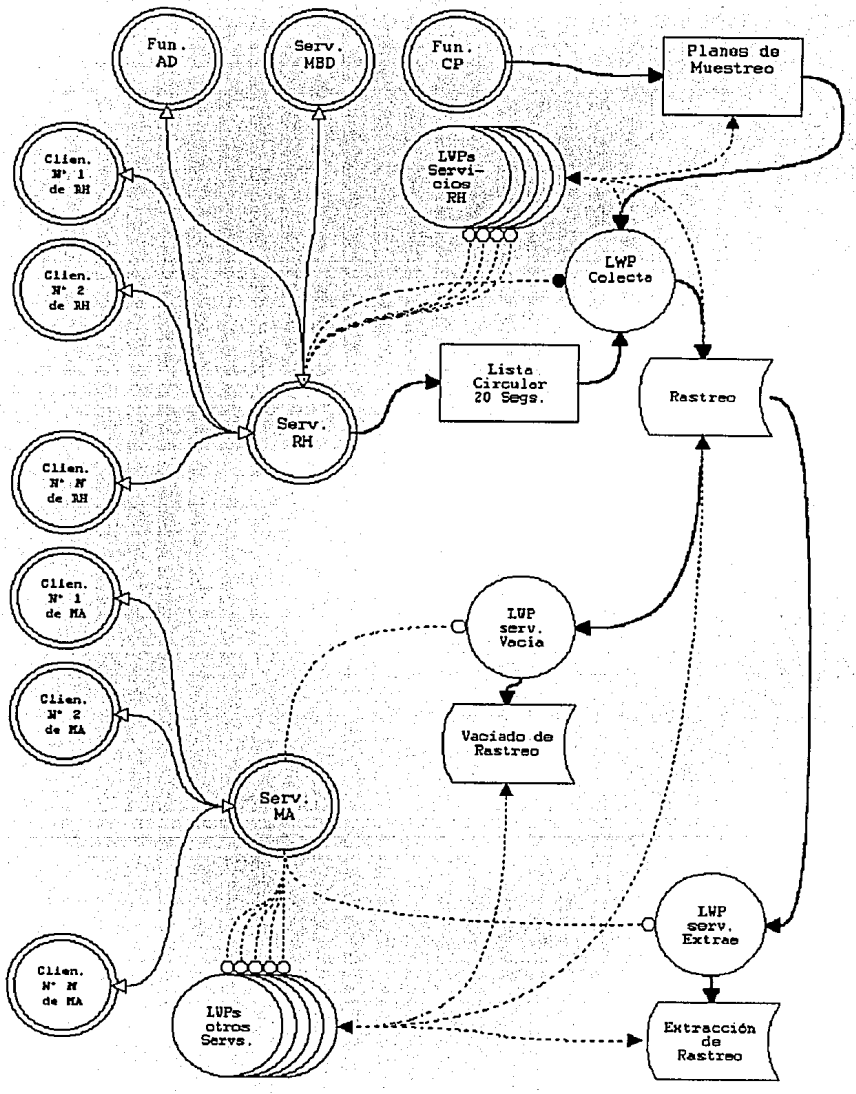
Fun. AD
Serv. MBD
Fun. CP
Planes de Muestreo
Clien. N° 1 de RH
LVPs Servi-cios RH
Clien. N° 2 de RH
LWP Colecta
Lista Circular 20 Segs.
Rastreo
Serv. RH
Clien. N° N de RH
Clien. N° 1 de MA
LWP serv. Vacia
Clien. N° 2 de MA
Vaciado de Rastreo
Serv. MA
LWP serv. Extrae
Clien. N° N de MA
LVPs otros Servs.
Extracción de Rastreo

Figura 5

En las figuras anteriores los círculos dobles representan procesos pesados, los círculos sencillos procesos ligeros, las líneas punteadas creaciones de procesos ligeros, las líneas gruesas flujo de datos, y las líneas delgadas llamadas a procedimientos remotos.

A continuación se muestran los seudocódigos de los procesos anteriormente descritos :

Programa

AD.

Argumentos

nodo.

Descripción

Simula la función de adquisición de datos *AD* que recibe hasta 5000 muestras por segundo del computador de adquisición, y envía las muestras al *nodo* donde se encuentra el servidor de Registro Histórico, *RH*.

Inicia ejecución

Llama a la función *clnt_create* con **parámetros** *nodo*, *RH_PROG*, *RH_VERS_1*, y *"tcp"* , para crear el handle del cliente, y pone **resultado** en *cl* ;

Asigna *semilla* **con** el tiempo de sistema, para función de numeros aleatorios ;

Asigna *folio* **con** 0 ;

Repite siempre

Asigna el tiempo inicial en *ini* **con** el tiempo del sistema ;

Incrementa *folio* **en** 1 y deposítalo en el primer valor de *segundo_val*[0], que en realidad no es una muestra ;

Asigna el segundo valor de *segundo_val*[1] **con** la hora actual en segundos tomada de *ini* , este valor tampoco es una muestra ;

Asigna las muestras que se van a enviar en *segundo_len* **con** un número al azar entre el máximo de muestras *MXMUESTRAS* entre 2 y *MXMUESTRAS* ;

Repite desde *k* = 2 (los primeros dos valores son el folio y la hora en segundos) **hasta** *segundo_len* , para generar cada una de las muestras ;

Asigna la dirección de la muestra *segundo_val*[*k*] **con** un número al azar entre 0 y 9 ;

Asigna las cuentas (valor binario de la muestra) de *segundo_val*[*k*] **con** un número al azar entre 0 y 4095 (12 bits) ;

Asigna la estampa en milisegundos de *segundo_val*[*k*] **con** un

número al azar entre los milisegundos de *ini* y *k* + 10 ;

Asigna el estatus de *segundo_val[k]* **con** 0 si un número al azar entre 0 y 100 es menor que 5 (5% de probabilidad), y **con** 0 de otra forma ;

Fin de Repite de *k* ;

Llama al procedimiento remoto *envía_muestras* con **parámetro** el arreglo de muestras *segundo_val* y pone **resultado** en *regresa* ;

Si *regresa* = *NULL* posiblemente se perdió la comunicación **entonces**

Llama a *clnt_destroy* para destruir el handle de cliente, con **parámetro** *cl* ;

Llama a la función *clnt_create* con **parámetros** *nodo, RH_PROG, RH_VERS_1, y "tcp"* , para crear de nuevo el handle del cliente, y pone **resultado** en *cl* ;

Si *regresa* = *NULL* de nuevo **entonces**
Imprime un diagnostico y **rompe** el ciclo ;

Si *regresa* = 0 , Significa que el servidor reporta que no pudo atender el servicio, **entonces**

Imprime un diagnostico y continua ;

/* De otro modo, significa que el servicio fue completado con éxito */

Asigna el tiempo final en *fin* **con** el tiempo del sistema ;

Si la diferencia entre *fin* y *ini* es mayor de un segundo **entonces**

Imprime un diagnóstico de los segundos que se perdieron y continua ;

Llama a *usleep* para dormir lo que falta (si es que falta) para un segundo, con **parámetro** *fin - ini* ;

Fin de Repite siempre

Llama a *clnt_destroy* para destruir el handle de cliente, con **parámetro** *cl* ;

Fin de ejecución

Programa

CL.

Argumentos

nodo , es el nodo donde se encuentran el servidor *RH* y el servidor *MA*.

servicio , es el nombre del servicio.

parámetros del servicio .

Descripción

El programa *CL* realiza llamados a los RPCs de los servidores *RH* y *MA* .

Inicia ejecución de *CL*

Llama a la función *clnt_create* con **parámetros** *nodo* , *RH_PROG* , *RH_VERS_1* y "*tcp*" , para crear el "handle" del cliente para *RH* **resultado** en *cl* ;

Si *cl* = *NULL* significa que NO pudo crear el "handle" de cliente para *RH* **entonces**

Imprime un diagnóstico y **aborta** la ejecución ;

En caso de que el *servicio* **sea**

"arranca" :

Llamar al procedimiento remoto *arranca* en el computador *nodo* ;

"detiene" :

Llamar al procedimiento remoto *detiene* en el computador *nodo* ;

"listar" :

Llamar al procedimiento remoto *listar* en el computador *nodo* ;

"cambia" :

Llamar al procedimiento remoto *cambia* en el computador *nodo* y con **parámetro** el nombre del plan a cambiar ;

Otro valor : Significa que el servicio puede ser del servidor *MA*

Llamar a *clnt_destroy* con **parámetro** *cl* , para destruir el handle de cliente para *RH* ;

Llamar a la función *clnt_create* con **parámetros** *nodo* , *MA_PROG* , *MA_VERS_1* y "*tcp*" , para crear el "handle" del cliente para *MA* **resultado** en *cl* ;

En caso de que el *servicio* **sea**

"vacía" :

Llamar al procedimiento remoto *vacía* en el computador *nodo* con **parámetro** el identificador del archivo receptor del vaciado ;

Otro valor : Significa que no existe ese servicio ni en *RH* ni en *MA*

Imprimir un diagnóstico y **Abortar** la ejecución de *CL* ;

Fin de casos

Fin de Casos

Llamar a *clnt_destroy* con **parámetro** *cl* , para destruir el handle de cliente para *RH* o para *MA* ;

Fin de ejecución de *CL*

Programa

RH.

Argumentos

ninguno.

Descripción

Servidor de la función de Registro Histórico *RH* que recibe las muestras de la función de Adquisición de Datos *AD* cuando ésta última llama al procedimiento remoto *envia_muestras* que es un servicio de *RH*, tal procedimiento deposita las muestras recibidas en una *lista_circular*. Además crea el LWP *colecta* que recoge las muestras de la *lista_circular* y las envía al archivo de *rastreo* de acuerdo al *plan* de muestreo, y posteriormente la información es explotada de el archivo *rastreo* por el servidor Manejador de Archivos *MA* .

Inicia ejecución de *RH*

Llama a la función *svctcp_create* con **parámetro** *RPC_ANYSOCK* para crear el servicio de "tcp" , y pone **resultado** en *transp* ;

Si *transp* = *NULL* significa que NO pudo crear el servicio de "tcp" **entonces**

Imprime un diagnóstico y **aborta** la ejecución ;

Si no puede registrar el servidor **llamando** a *svc_register* con **parámetros** *transp, RH_PROG, RH_VERS_1, rh_prog_1* y *IPPROTO_TCP* **entonces**

Imprime un diagnóstico y **aborta** la ejecución ;

Llama a *svc_run* sin **parámetros** , y esta función no debe de regresar porque tiene un ciclo infinito, solamente regresa cuando hay algún error ;

Imprime un diagnóstico y **aborta** la ejecución ;

Fin de ejecución de *RH*

Función

svc_run en *RH*

Parámetros

ninguno

Descripción

La función *svc_run* es llamada por el programa *RH* y tiene como objetivo hacer las inicializaciones necesarias del servidor y entrar en un ciclo infinito donde espera por medio de la función *select* , a que algún cliente requiera llamar a un procedimiento remoto (servicio) de *RH*. Además no sólo llama al RPC sino que crea un LWP por cada llamada.

Inicia ejecución de *svc_run* en *RH*

Llama a *carga_planes* **sin parámetros** , para llenar la memoria de planes de muestreo ;

Llama a *pod_setmaxpri* con **parámetro** *MAXPRIO* , para crearse a si mismo como un hilo de activación (LWP) con prioridad *MAXPRIO* ;

Llama a *lwp_setstkcache* con **parámetros** *1024* y *MXSERVICIOS + 2* para definir el tamaño de las pilas de los LWPs y cuantas de éstas se reservan, dos para el LWP principal y el *colecta* y una por cada servicio ;

Llama a *mon_create* y a *cv_create* para crear el monitor *sobre_lista* y las variables de condición *no_vacia* y *no_llena* sobre el mismo monitor.

Llama a *lwp_create* para crear el LWP *colecta* con prioridad *MAXPRIO-1* ;

Repite para siempre

En caso de que al **Llamar** a *select* , que es la función que revisa si hay un servicio pendiente y si no se pone a dormir **sea**

-1 : Puede ser que hay un error

Si el error es recuperable **entonces**
continuar el ciclo ;
De otro modo
Imprimir un diagnóstico y **Termina** la ejecución de *svc_run* ;

0 : Significa que no hay servicio después de un tiempo

continuar en el ciclo ;

Otro valor : Significa que hay al menos un servicio pendiente

Llamar a *lwp_create* para crear un LWP con *svc_getreqset* , que es la rutina que llama al procedimiento remoto y regresa el resultado al programa que solicitó el servicio , con prioridad *MAXPRIO* (que si es no necesaria tanta , dentro del LWP se cambia la prioridad) ;

Llama a *lwp_yield* para ceder la ejecución al LWP recién creado ;

Fin de Casos

Fin de repite para siempre

Fin de ejecución de *svc_run* en *RH*

Función

envia_muestras en *RH*

Parámetros

segundo que tiene hasta 5000 muestras (*dirección*, *cuentas*, *estampa* en milisegundos y *estatus*) mas el *folio* y la *hora* en segundos.

Descripción

La función *envia_muestras* es llamada desde el LWP creado con *svc_getreqset* por la función *svc_run* y tiene como objetivo tomar el parámetro *segundo* y depositarlo en la *lista_circular* , sincronizado por medio del monitor *sobre_lista* con el LWP *colecta*. El LWP de *envia_muestras* conserva la prioridad *MAXPRIO* con que fue creado, dada la importancia de que todas las muestras se registren en la *lista_circular*.

Inicia ejecución de *envia_muestras* en *RH*

Llama ala macro *MONITOR* con **parámetro** *sobre_lista* para proteger toda la función con tal monitor ;

Si *registrar* = 0 Significa que no está arrancado el registro **entonces**

Termina la ejecución de *envia_muestras* **regresando** en el resultado que el registro no está arrancado ;

Repite mientras *cuantos* = *TMCIRCULAR* Es decir mientras que la lista este llena

Llamar a *cv_wait* para esperar (Se bloquea el LWP) por la condición *no_llena* ;

Fin de Repite mientras

Asignar el siguiente elemento de la *lista_circular* **con** el valor de *segundo* (las 5000 muestras) ;

Incrementar el tamaño de la *lista_circular* en *cuantos* y el apuntador al *rabo* de la lista ;

Llamar a la función *cv_broadcast* para despertar a los procesos que esperan por la condición *no_vacia* ;

Fin de ejecución de *envia_muestras* en *RH*

Función

colecta en *RH*

Parámetros

ninguno.

Descripción

La función *colecta* es el código que ejecuta un LWP creado por la función *svc_run* , este LWP se crea al inicio de la ejecución de *svc_run* y es una función que se cicla y permanece compitiendo por el CPU con prioridad *MAXPRIO* - 1 durante el resto de la ejecución del servidor *RH* . La función *colecta* toma elementos de la *lista_circular* llamando a *saca* , y de acuerdo al *plan* de muestreo , deposita las muestras en el archivo de *rastreo* .

Inicia ejecución de *colecta* en *RH*

Asigna *folio_anterior* **con** 0 ;

Repite para siempre

Llama a la función *saca* con **parámetro** *segundo* para recoger un elemento (hasta 5000 muestras) de la *lista_circular* ;

Si el folio en *segundo* (*segundo_val*[0]) es mayor que el *folio_anterior* + 1 **entonces**

Imprimir un diagnóstico de los folios que se perdieron ;

Repite desde *k* = 2 (los primeros dos valores son el folio y la hora en segundos) **hasta** *segundo_len* (el numero de muestras en el *segundo*) , para recorrer cada una de las muestras ;

Si de acuerdo al *plan* de muestreo la *muestra* de *segundo* se debe registrar **entonces**
Escribir en el archivo *rastreo* la hora que se encuentra en el segundo valor de *segundo* (*segundo_val*[1]) y la muestra que está en *segundo_val*[*k*] ;

Fin de Repite de *k* ;

Asigna *folio_anterior* **con** el folio de *segundo* (*segundo_val*[0]) ;

Fin de repite para siempre
Fin de ejecución de *colecta* en *RH*

Función

detiene en *RH*

Parámetros

ninguno.

Descripción

La función *detiene* es llamada por la función *svc_getregset* , que es el código que ejecuta un LWP creado por la función *svc_run* para atender el servicio de detener el registro en el archivo *rastreo*. Este LWP conserva la prioridad *MAXPRIO* , puesto que este servicio se debe atender inmediatamente.

Inicia ejecución de *detiene* en *RH*

Si *registrar* = 0 **entonces**
Termina la ejecución de *detiene* **regresando** en el resultado que el registro ya estaba detenido ;

Asignar *registrar* **con** 0 ;

Fin de ejecución de *detiene* en *RH*

Función

cambia en *RH*

Parámetros

nombre , nombre del plan de muestreo a cambiar.

Descripción

La función *cambia* es llamada por la función *svc_getreqset* , que es el código que ejecuta un LWP creado por la función *svc_run* para atender el servicio de cambiar el plan de muestreo actual *plan_actual* . El LWP conserva la prioridad *MAXPRIO* con que fue creado, debido a que el servicio se debe ejecutar inmediatamente.

Inicia ejecución de *cambia* en *RH*

Si *registrar* = 1 Significa que el registro está activo **entonces** **Termina** la ejecución de *cambia* **regresando** en el resultado que el registro no está detenido ;

Si *nombre* está en la memoria de planes de muestreo *plan* **entonces** **Asignar** *plan_actual* **con** *nombre* ;

De otro modo
Termina la ejecución de *cambia* **regresando** en el resultado que el *nombre* de plan no existe ;

Fin de ejecución de *cambia* en *RH*

Función

listar en *RH*

Parámetros

ninguno.

Descripción

La función *listar* es llamada por la función *svc_getreqset* , que es el código que ejecuta un LWP creado por la función *svc_run* para atender el servicio de listar los planes de muestreo que existen en la memoria. Este LWP cambia su prioridad a *MINPRIO* puesto que no es necesario que se ejecute este servicio inmediatamente.

Inicia ejecución de *listar* en *RH*

Llama a la función *lwp_setpri* con **parámetro** *MINPRIO* para cambiarse a si mismo la prioridad ;

Copia los nombres de *plan* de muestreo **en** el arreglo *resultado* ;

Fin de ejecución de *listar* en *RH* **regresando** *resultado* en el nombre de la función ;

Función

saca en *RH*

Parámetros

segundo , es un parámetro por dirección y tiene hasta 5000 muestras.

Descripción

La función *saca* es llamada por la función *colecta* cada vez que necesita extraer un elemento de la *lista_circular* .

Inicia ejecución de *saca* en *RH*

Llama ala macro *MONITOR* con **parámetro** *sobre_lista* para proteger toda la función con tal monitor ;

Repite mientras *cuantos* = 0 Es decir mientras que la lista este llena ;

Llamar a *cv_wait* para esperar (Se bloquea el LWP) por la condición *no_vacia* ;

Fin de Repite mientras

Copia el elemento de la *lista_circular* del *frente* **en** *segundo* ;

Decrementar el tamaño de la *lista_circular* en *cuantos* e **Incrementar** el apuntador al *frente* de la lista ;

Llamar a la función *cv_broadcast* para despertar a los procesos que esperan por la condición *no_llena* ;

Fin de ejecución de *saca*

Programa

MA.

Argumentos

ninguno.

Descripción

Servidor de Manejo de Archivos *MA* para la función de Registro Histórico. Este servidor toma los datos depositados por el servidor de Registro Histórico *RH* en el archivo *rastreo* y atiende los servicios de extracción y vaciado de archivos.

Inicia ejecución de *MA*

Llama a la función *svctcp_create* con **parámetro** *RPC_ANYSOCK* para crear el servicio de "tcp" , y pone **resultado** en *transp* ;

Si *transp* = *NULL* significa que NO pudo crear el servicio de "tcp" **entonces**

Imprime un diagnóstico y **aborta** la ejecución ;

Si no puede registrar el servidor **llamando** a *svc_register* con **parámetros** *transp, MA_PROG, MA_VERS_1, ma_prog_1* y *IPPROTO_TCP* **entonces**

Imprime un diagnóstico y **aborta** la ejecución ;

Llama a *svc_run* sin **parámetros** , y esta función no debe de regresar porque tiene un ciclo infinito, solamente regresa cuando hay algún error ;

Imprime un diagnóstico y **aborta** la ejecución ;

Fin de ejecución de *RH*

Función

svc_run en *RH*

Parámetros

ninguno

Descripción

La función *svc_run* es llamada por el programa *MA* y tiene como objetivo hacer las inicializaciones necesarias del servidor y entrar en un ciclo infinito donde espera por medio de la función *select* , a que algún cliente requiera llamar a un procedimiento remoto (servicio) de *MA*. En el caso de los servicios de *MA* que vacían o realizan extracciones del archivo de *rastreo* estos servicios no regresan al programa llamador hasta que el servicio se concluye, si no que el procedimiento de servicio crea un LWP y regresa un resultado que significa que el servicio está en proceso, lo anterior permite que se puedan atender varios servicios que tienen duración larga sin bloquear la recepción de peticiones de servicio.

Inicia ejecución de *svc_run* en *MA*

Llama a *pod_setmaxpri* con **parámetro** *MAXPRIO* , para crearse a si mismo como un hilo de activación (LWP) con prioridad *MAXPRIO* ;

Llama a *lwp_setstkcache* con **parámetros** *8162* y *MXSERVICIOS* para definir el tamaño de las pilas de los LWPs y cuantas de éstas se reservan, una por cada servicio que llama a un LWP ;

Llama a *mon_create* para crear el monitor *atómico* para proteger operaciones atómicas ;

Repite para siempre

En caso de que al **Llamar** a *select* , que es la función que revisa si hay un servicio pendiente y si no se pone a dormir **sea**
-1 : Puede ser que hay un error

Si el error es recuperable **entonces**
continuar el ciclo ;
De otro modo
Imprimir un diagnóstico y **Termina** la ejecución de *svc_run* ;

0 : Significa que no hay servicio después de un tiempo

continuar en el ciclo ;

Otro valor : Significa que hay al menos un servicio pendiente

Llamar a *svc_getreqset* , que es la rutina que llama al procedimiento remoto y regresa el resultado al programa que solicitó el servicio ;

Fin de Casos

Fin de repite para siempre

Fin de ejecución de *svc_run* en *MA*

Función

vacía en *MA*

Parámetros

nombre Es el nombre del archivo que se va a crear.

Descripción

La función *vacía* es llamada *svc_getreqset* , y su objetivo es crear un LWP con *MINPRIO* para que haga el vaciado y regresar al cliente un resultado de que el vaciado está en proceso.

Inicia ejecución de *vacía* en *MA*

Llama ala función *lwp_create* con **parámetros** *proceso_vacía* , *MINPRIO* y el *nombre* del archivo receptor, para crear el LWP que realiza la copia ;

Fin de ejecución de *vacía* en *MA* **regresando** que el vaciado está en proceso.

Función

proceso_vacía en *MA*

Parámetros

nombre , identificador del archivo receptor del vaciado.

Descripción

La función *proceso_vacía* es el código que ejecuta un LWP creado por la función *vacía* , Este LWP se cambia la prioridad *MINPRIO* y realiza la copia del archivo *rastreo* al archivo que tiene identificador externo *nombre*.

Inicia ejecución de *proceso_vacía* en *MA*

Llama a la función *lwp_sleep* con **parámetro** *chanza* (que fue inicializado en 100 milisegundos) para dar oportunidad que otros LWPs sean creados y que otros servicios sean atendidos antes de empezar la copia ;

Repite mientras no se acabe el archivo *rastreo*

Lee un buffer del archivo *rastreo* ;

Escribe un buffer en el archivo con identificador *nombre* ;

Fin de repite mientras

Si la copia falló **entonces**
Imprime un diagnóstico y **Termina** la ejecución de *proceso_vacía* ;

Fin de ejecución de *proceso_vacía* en *MA*

Evaluación del Servidor

Se realizaron dos tipos de evaluación del desempeño de la función de registro histórico, una sobre el servidor RH, y otra sobre el servidor MA. Las pruebas se realizaron en computadoras "Spark Station" de Sun, conectadas en red Ethernet y bajo Sun OS, como lo muestra la figura 6.

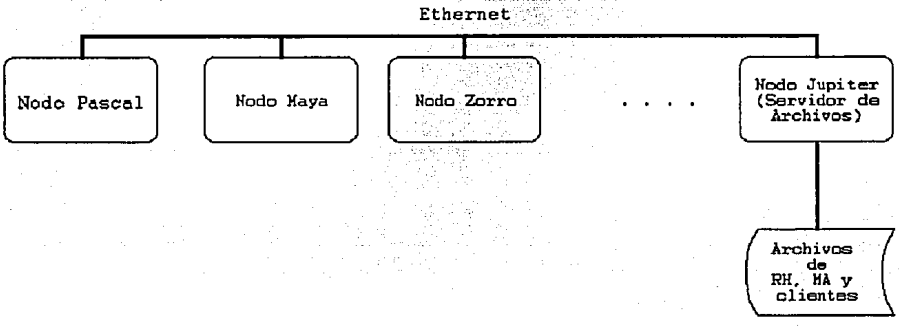

Figura 6

Referente al servidor RH, se evaluaron los tiempos que tarda en contestar al proceso de adquisición desde diferentes nodos, obteniéndose los siguientes resultados, donde los períodos están dados en segundos y los tiempos en microsegundos :

Caso 1 : AD en el nodo Pascal y servidor de RH en el nodo Pascal

Período	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio
10	50907	28762	37596
50	50976	28477	40022
100	55850	23608	41737
1000	149418	23127	41093

Caso 2 : AD en el nodo Maya y servidor de RH en el nodo Pascal

Período	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio
10	46440	22139	32066
50	780232	22491	62400
100	40520	22695	33499
1000	217648	22306	44073

Caso 3 : AD en el nodo Zorro y servidor de RH en el nodo Pascal

Período	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio
10	97716	22213	41212
50	223148	20905	61523
100	216337	20729	45387
1000 *	No se pudo realizar.		

Caso 4 : AD en el nodo Maya y servidor de RH en el nodo Maya

Período	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio
10	79346	46235	67271
50	78669	36149	61746
100	83897	44014	62356
1000	121153	43381	63863

Caso 5 : AD en el nodo Pascal y servidor de RH en el nodo Maya

Período	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio
10	142887	30792	48729
50	22359	29766	60924
100	234040	30865	60295
1000	269748	29600	59614

Caso 6 : AD en el nodo Zorro y servidor de RH en el nodo Maya

Período	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio
10	119776	30671	49666
50	226980	29788	47619
100	231921	30543	57635
1000	274048	29559	58616(*) 14 fallas

Caso 7 : AD en el nodo Pascal y servidor de RH en el nodo Zorro

Período	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio
10	215501	23503	54665
50	217837	24313	59068
100	224448	23003	51309
1000	989762	9628	66283(*) 102 fallas

Caso 8 : AD en el nodo Zorro y servidor de RH en el nodo Zorro

Período	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio
10	54286	33878	41963
50	66524	29153	42430
100	127343	28611	44076
1000	212652	26242	45511

Caso 9 : AD en el nodo Maya y servidor de RH en el nodo Zorro

Período	Tiempo Máximo	Tiempo Mínimo	Tiempo Promedio
10	221904	27402	58157
50	994423	26219	101347
100	830925	266	67140
1000	934965	25082	6174

Las gráficas de las figuras 7,8 y 9 muestran el comportamiento para el período de 1000 segundos en todos los casos.

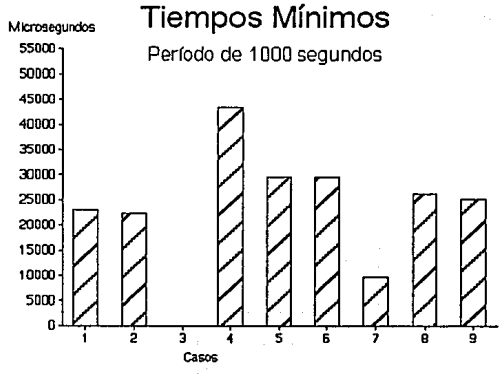

Figura 7

Figura 8

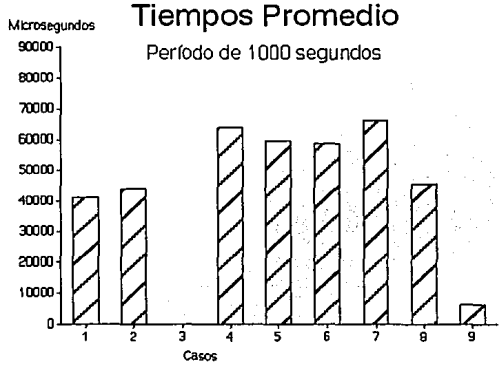

Figura 9

Con respecto al tiempo que tarda RH en escribir en el archivo de *rastreo*, se observó que con el máximo régimen de muestreo se tarda 48 minutos por cada Megabyte que escribe. Por lo tanto un archivo de 12 horas tendría 15 Megabytes. Se evaluó también la resistencia de los vaciados del archivo *rastreo* , es decir si MA soporta un numero de clientes haciendo la operación mas costosa en tiempo y espacio, que es al de vaciado del archivo *rastreo*. Se esperaban en esta prueba tres tipos de resultado :

a) El *rastreo* se copió al *vaciado* íntegramente y el cliente recibió notificación de que el vaciado estaba en proceso.

Llamamos a este resultado CORRECTO.

b) El *rastreo* se copió al *vaciado* íntegramente pero el cliente NO fue notificado de que el vaciado está en proceso, de hecho el cliente no recibe respuesta del servidor. Llamamos a este resultado COMPLETO.

c) El *rastreo* se copió al *vaciado* a medias y el cliente NO fue notificado de que el vaciado está en proceso. Llamamos a este resultado INCOMPLETO.
d) El *rastreo* NO se copio al *vaciado* , pero el cliente recibió notificación de que el vaciado está en proceso. Llamamos a este resultado INCORRECTO.

e) El cliente ni siquiera pudo establecer comunicación con el servidor. Llamamos a este resultado DESCONECTADO.

f) El proceso de adquisición reporta pérdida de datos, debido a que RH no puede vaciar su lista circular con suficiente rapidez. Llamamos a este resultado FALLA, independientemente de si se realizó o no el vaciado.

En la primera versión del servidor, donde se tenían a RH y MA en un solo proceso pesado, se observaron resultados CORRECTOS en vaciados hasta de 300 Kbytes. En seguida, en archivos de mas de 300 Kbytes y hasta 1.6 Megabytes, se observaron resultados CORRECTOS en un 38% de los casos y COMPLETOS en un 62%, después de 1.6 Megabytes se observó un 12% de CORRECTOS, un 68% de COMPLETOS y un 50% de FALLAS y DESCONECTADOS. Los resultados anteriores no fueron alentadores y esto fue lo que llevo a dividir el servidor en dos RH y MA, esto permite que los vaciados de archivos no interfieran con el tiempo que el proceso RH tiene asignado por el Sistema Operativo, además de sincronizar el acceso al archivo *rastreo* como un escritor y muchos lectores.

Una vez realizadas las correcciones mencionadas anteriormente, se hicieron pruebas de ocho peticiones de vaciado de archivo simultaneas con diferentes tamaños de archivo de *rastreo* y el resultado fue el siguiente :

TAMAÑO de *rastreo*	RESULTADO
200 Kb	8 Correctos
650 Kb	8 Correctos
1.4 Mb	3 Correctos y 5 Completos
1.7 Mb	2 Correctos y 6 Completos
1.8 Mb	7 Correctos y 1 Completo

2 Mb	6 Correctos y 2 Completos
2.86 Mb	4 Correctos y 4 Completos
3 Mb	8 Correctos
3.5 Mb	5 Correctos y 3 Completos
4 Mb	8 Correctos
5 Mb	1 Correcto y 7 Completos

De las evaluaciones de los servidores RH y MA podemos observar lo siguiente :

1) El tiempo máximo para transferir un segundo de muestras fue de .989762 segundos, en el caso 7, que es todavía menor de un segundo.

2) A pesar de que en los casos 6 y 7, hubo fallas (desconexiones), el servidor se pudo recuperar y realizar las transferencias satisfactoriamente.

3) En ninguna de las pruebas realizadas a MA hubo errores en las copias.

4) En la prueba de MA con *rastreo* de 4 Mb., el cliente siempre recibió notificación de que el vaciado está en proceso, a pesar de que en pruebas con tamaños mas pequeños (1.4 Mb., 1.7 Mb., 1.8 Mb. etc...) no se presentó este comportamiento.

V CONCLUSIONES

De la evaluación hecha en el capítulo anterior se desprenden las siguientes conclusiones :

1) La respuesta de los RPCs no depende de la velocidad de los procesadores, sino mas bien de la carga que existe en la red.

2) Debido a que el sistema en el que se hicieron las pruebas no es un sistema dedicado al control de procesos, sino mas bien un sistema de tiempo compartido el tráfico en la red no es predecible.

3) A pesar de lo expuesto en el punto anterior, el servidor soportó el peor de los casos que es con un plan de muestreo donde todas las variables aparecen y cada una con una frecuencia muestreo de 2 milisegundos.

4) Las copias de archivos se efectuaron con problemas menores y relacionados sólo con la respuesta del servidor al cliente, pero soportando copias hasta de 40 Megabytes.

5) Se comprobó que el esquema de Llamadas a Procedimientos Remotos puede funcionar para requisitos de tiempo como los de la función de Registro Histórico.

6) Se estableció que para este tipo de aplicaciones es muy conveniente usar un protocolo orientado a conexión en los RPCs, para garantizar que los mensajes se reciban, pero en ocasiones no se puede esperar una respuesta a una llamada que tarde mucho y se debe dejar que el servicio se realice en el servidor.

7) Se comprobó que los procesos de peso ligero son de gran utilidad en este tipo de sistemas porque es imperceptible el retardo debido a la creación, bloqueo, y destrucción de los LWPs.

8) También se comprobó que los monitores y las variables de condición, son mecanismos ideales para sincronizar actividades de los LWPs , debido a que no necesitan del "kernel" para su funcionamiento.

9) Se comprobó también la utilidad de las jerarquías de prioridades en los LWPs, que permiten clasificar los procesos por importancia, y así asignar el tiempo a cada uno.

10) Para el desarrollo del prototipo se utilizó la norma POSIX.1, la cual está disponible actualmente en muchos sistemas. En nuestro caso, el sistema operativo que se usó fue Sun OS, que soporta el estandard. Con esto se logra un alto grado de transportabilidad del prototipo.

11) Se utilizaron primitivas de Sun OS equivalentes a las existentes en las extensiones de tiempo real y de procesos de peso ligero de POSIX, tales como monitores (o Mutex como se denominan en POSIX), variables de condición, prioridades, relojes en Microsegundos, y creación dinámica de LWPs. Estos estándares no están tan aceptados debido que hasta ahora sólo son bosquejos de estándares, pero que cada vez se ofrecen con mas frecuencia por proveedores. La conversión de tales primitivas a código POSIX es casi uno a uno.

12) Se utilizó el sistema ONC de Sun y AT&T, para RPCs dado que es el mas usado, el que se tenía disponible y las modificaciones para hacer lo mismo en el sistema de HP/Apollo involucra cambios mas de forma que de concepto.

Trabajos Futuros

Se pueden mencionar dos líneas de investigación a futuro, una siguiendo los paradigmas que se utilizaron en este trabajo, y otra intentando utilizar otros modelos o modificaciones a estos.

Relacionado con los paradigmas de modelo cliente/servidor, llamadas a procedimientos remotos y procesos de peso ligero, se puede pensar en las siguientes mejoras :

1) Realizar pruebas con protocolos de red que tengan mas facilidades de tiempo real, tales como el cumplimiento de metas de tiempo en los mensajes, asignación de prioridades, circuitos virtuales o métodos de reservaciones. Asimismo con redes que ofrezcan mas seguridad del despacho de mensajes que Ethernet.

2) De la misma manera que las muestras en RH son dirigidas a una lista circular, es conveniente que el despacho desde AD También sea desde una lista circular para amortiguar los efectos de la red en el envío, y por supuesto el uso de procesos de peso ligero dentro de AD para compartir la memoria común.

3) Realizar un prototipo del resto de las funciones del SADRE bajo los mismos paradigmas y evaluar los resultados en un sistema dedicado.

4) Modificar los esquemas de copia a archivos masivo, de tal manera que los procesos de copiado tomen descansos periódicos para permitir que el servidor atienda a mas clientes. Lo anterior es un problema que no se presentó en la última versión del prototipo, pero puede suceder quizá con procesadores mas lentos.

5) Caracterizar las funciones de un SADRE, para determinar cuales pueden ser realizadas con los paradigmas anteriores.

6) Estudiar el orden de complejidad en tiempo y en numero de mensajes del servidor.

Con respecto a la segunda línea se puede sugerir lo siguiente :

1) Realizar el mismo servidor, pero usando el modelo orientado a objetos en lugar del esquema cliente/servidor.

2) Realizar el mismo servidor pero usando el sistema NCS de HP/Apollo para los RPCs.

3) Utilizar lenguajes de alto nivel que contienen facilidades de tiempo real, manejo de procesos y mecanismos de intercomunicación, tales como ADA.

VI REFERENCIAS BIBLIOGRÁFICAS

Arvind 91

Local Area Network Architecture for Communication in Distributed Real Time Systems, Arvind, K.; Ramamrithan, Krithi; Stankovic, John A.: Real time Systems v3 n2 May 1991.

Bloomer 92

Power programming with RPC, John Bloomer, O'Reilly & Associates 1992.

Corbin 91

The Art of Distributed Applications, Programming Techniques for Remote Procedure Calls. Springer-Verlag 1991.

Goscinski 91

Distributed Operating Systems, Andrezej Goscinski, Addison-Wesley 1991.

IEEE 93 a

Standards Project P1003.4/D14, Draft Standard for Information Technology Portable Operating Systems Interface, System Application Program Interface (APE) Real Time Extension, Technical Committee on Operating Systems Application Environments of The IEEE computer Society, March 1993.

IEEE 93 b

Standards Project P1003.4 Draft 7, Draft Standard for Information Technology Portable Operating Systems Interface (POSIX), System Application Program Interface (API) Threads Extension, Technical Committee on Operating Systems Application Environments of The IEEE computer Society, April 1993.

IIE 93 a

Especificación de Requisitos Funcionales de la IHM Para el Sistema de Control Distribuido de la Central de Ciclo Combinado de Gómez Palacio Dgo., Instituto de Investigaciones Eléctricas, Departamento de Automatización de Procesos, 21 de Julio de 1993, Cuernavaca Morelos.

IIE 93 b

Especificación de Requisitos de Software de la Interfase Hombre-Máquina para la Central de Ciclo Combinado de Gómez Palacio Dgo., Instituto de Investigaciones Eléctricas, Departamento de Automatización de Procesos, 23 de Julio de 1993, Cuernavaca Morelos.

IIE 93 c

Anexo Técnico del Proyecto 5147 "Sistemas de Adquisición de Datos y Registro de Eventos con Nuevas Tecnologías", Instituto de Investigaciones Eléctricas, Departamento de Automatización de Procesos, 15 de Febrero de 1993, Cuernavaca Morelos.

Kernighan 78

The C Programming Language, Brian W. Kernighan, Dennis M. Ritchie, Prentice Hall 1978.

Kernighan 84

The UNIX Programming Environment, Kernighan, Pike, Prentice Hall 1984.

Khanna 92

Real Time Scheduling in SunOS 5.0, Khanna, S.; Sebree, M.; Zolnowsky, J., USENIX Winter '92.

Kleiman 92

Writing multithread Code in Solaris Steven Kleiman, Bart Smaalders, Dan Stein, Devang Shah, Kernel and Application Threads Architecture, six technical papers, SunSoft 1992, Sun MicroSystems Inc. Mountain View CA.

Kleiman 92

Symmetric Multriprocessing in Solaris 2.0, Steven Kleiman, Jim Voll, Joe Eycholt, Anil Shivalingiah, Dock Williams, mark Smith, Steve Berton, Glenn Skinner, Kernel and Application Threads Architecture, six technical papers, SunSoft 1992, Sun MicroSystems Inc. Mountain View CA.

Lewine 92

POSIX Programmer's Guide, Donald Lewine, O'Reilly & Associates 1992.

Mullender 89

Distributed Systems, Sape Mullender, ACM Press 1989.

Northcutt 87

Mechanisms for Reliable Distributed Real-Time Operating Systems, Academic Press 1989.

Powell 92

Sun OS Multi-Thread Architecture, M. L. Powell, S. R. Kleiman, D. Shah, D. Stern, M. Weeks, Kernel and Application Threads Architecture, six technical papers, SunSoft 1992, Sun MicroSystems Inc. Mountain View CA.

Scientific 90

C-Linda Reference Manual, Scientific Computing Associates, 1990.

Stankovic 88

Misconceptions About Real-Time Computing, A Serious Problem for Next Generation Systems, John A Stankovic, IEEE COMPUTER, October 1988.

Stankovic 90

What is Predictability for Real-Time Systems ?, John A Stankovic, Real Time Systems 1990.

Stankovic 92 a

Resource Allocation in Real-Time Systems, John Stankovic, Real-time Systems 1992.

Stankovic 92 b

Real Time Computing, John A. Stankovic, Byte Aug. 1992. Stein, D. 92

Stein, D. 92

Implementation of Lightweight Threads, D. Stein, D. Shah, Summer 92 USENIX June 8 to June 12 1992, San Antonio TX.

Stein, R. 92

Real Time POSIX, Richard Marlon Stein, Byte Aug. 1992.

Stevens 90

UNIX Network Programming, W. Richard Stevens, Prentice Hall 1990.

Sun 89

C Programming Guide. Sun Microsystems Inc. 1989.

Sun 91

Desktop Spark, Deskset Reference Guide, Sun Microsystems Inc. 1991.

Sun 92

Solaris Kernel and Application Threads Architecture, six technical papers, SunSoft 1992, Sun MicroSystems Inc.

Sun 92

Multithreading and Real-Time in Solaris, Terms and Concepts, SunSoft 1992, Sun Microsystems Inc.