DISEÑO Y EVALUACION DE UN SERVIDOR DE RECURSOS PARA UNA RED LOCAL*

Francisco Aurtenechea O.
Ignacio Casas R.
Roberto Cumsille U.
Marco Kútulas. P

Pontificia Universidad Católica de Chile
Depto. de Ciencia de la Computación
Casilla 6177, Santiago, Chile
FAX: (562) 5524054 email: fa@puc.cl

RESUMEN

En este trabajo se presenta el diseño y análisis de un sistema de acceso transparente a servicios distribuídos en una red local Ethernet de la Universidad Católica de Chile. El sistema está soportado por un conjunto de primitivas de comunicación de alto nivel, cuyo objetivo es facilitar el desarrollo de sistemas distribuídos, especialmente en relación al acceso transparente a recursos de una red local. El sistema consta actualmente de una interfaz de comunicación entre procesos remotos, tanto para mini-computadores Unix como micro-computadores PC-DOS, y una interfaz de usuario para micro-computadores PC-DOS. Ambas interfaces contienen primitivas que pueden ser invocadas desde el lenguaje C. Se incluye un análisis del desempeño del sistema en el acceso a aplicaciones remotas, y se compara con el acceso local.

I. INTRODUCCION

En ambientes universitarios es muy común encontrar una gran variedad de equipamiento computacional. La tendencia es hacia la provisión de herramientas que faciliten la interconexión de estos equipos mediante el uso de redes locales, a diferentes niveles de servicio. Entre estas herramientas se incluyen tanto hardware estándar de conexión (p.ej.: tarjetas Ethernet) como mecanismos simples de intercambio de información vía software (p.ej.: login remoto, correo electrónico, transferencia de archivos).

Sin embargo, la programación de aplicaciónes más sofisticadas, especialmente aquellas que involucran procesamiento distribuído [Che88, Tan85] requieren de un alto grado de conocimiento de los sistemas operativos en los cuales se requiere correr la aplicación. Además, el programador debe involucrarse con estructuras de datos y primitivas de comunicacíon de bajo nivel que son distractoras del objetivo que se persigue al construir una aplicación. Normalmente, para permitir la conexión estre dos

* Este trabajo ha sido financiado por FONDECYT, proyecto 0187/91

o más procesos remotos o el envío de mensajes entre ellos, es necesario escribir un trozo de código no trivial compuesto por llamados a rutinas del sistema operativo, p. ej.: mediante el uso de una biblioteca de sockets [Exc88].

Por otro lado, en la medida que las redes se expanden, evolucionan y la demanda por su uso aumenta, es importante conocer las limitaciones en el desempeño que imponen a las aplicaciones de red. Por ende, los datos de estudios de desempeño son una evidencia importante para estrategias de diseño y planificación de capacidad.

El propósito de este trabajo es proporcionar un medio ambiente de software para el programador de aplicaciones en el ámbito de sistemas distribuídos, donde coexisten sistemas heterogéneos, incluyendo sistemas multiusuarios y micro-computadores conectados en una red local Ethernet [Met76]. El sistema está orientado hacia la provisión de una interfaz de comunicación homogénea, de alto nivel, compuesta por un conjunto de primitivas de comunicación entre procesos que permitan hacer transparente el procedimiento de comunicación particular de cada sistema. El análisis de desempeño del sistema, mediante una herramienta prototipo, se presenta también como un objetivo de este trabajo, y se considera a futuro la provisión de herramientas de desempeño de carácter general para redes locales.

Como base de desarrollo, se diseñó un núcleo de comunicación compuesto por un conjunto de primitivas de enlace entre procesos remotos. Estas permiten esconder al programador los problemas inherentes al establecimiento de sesiones entre procesos de la red, presentando un nivel de abstracción mayor que las herramientas tradicionales de comunicación entre procesos. De este modo, se pretende simplificar el desarrollo de sistemas de acceso a recursos distribuídos, liberando al programador de los detalles de bajo nivel en la comunicación entre procesos de una red. El objetivo es incrementar la productividad del programador y potenciar la gama de posibilidades para el uso de los recursos que ofrece la red, especialmente a microcomputadores conectados a ésta.

El desarrollo de estas rutinas se sustenta en el uso de las herramientas proporcionadas por la biblioteca estándar Socket Library [Exc88], y que están disponibles actualmente en la mayoría de los computadores de la red. A nivel de comunicación en la red, se utilizan protocolos de comunicación estándar tal como TCP/IP. Sin embargo, estas facilidades son transparentes para el programador ya que

operan en los niveles de transporte y de red dentro del modelo de red OSI [Iso81], definido para arquitecturas de comunicación abiertas.

Mediante el núcleo de comunicación desarrollado se contruyó un sistema de acceso transparente a los recursos disponibles en distintos equipos Unix de la red, bajo el esquema **cliente/servidor** [ver Tan85], incorporando como clientes tanto a equipos Unix como equipos MS-DOS. En beneficio de la simplicidad en la provisión de los numerosos potenciales servicios de la red, en cada nodo se incorpora un único **servidor de procesos** que activa a un proceso específico cuando un cliente solicita un servicio.

El sistema fue concebido, fundamentalmente, para ser usado en un ambiente universitario, donde los requerimientos computacionales, además de que normalmente no involucran grandes volúmenes de datos, son del más variado tipo: compilación, acceso a sistemas de bases de datos, uso de sofware estadístico, edición de documentos, impresión de archivos, etc.

Los clientes Unix proveen una interfaz de usuario simple, en base a un único comando. En los micro-computadores la interfaz de usuario está compuesta por un conjunto de rutinas gráficas que permiten desarrollar un esquema de "pull down" menús y ventanas. Este sistema permite tener varias aplicaciones remotas concurrentes en la pantalla del PC, cada una desplegando su salida en una ventana independiente y en forma transparente a la red.

Finalmente, aprovechando las herramientas construídas, se diseñó un prototipo de sistema de evaluación del desempeño en el acceso a servicios remotos basados en el modelo cliente/servidor. Se incluyen resultados comparativos de tiempos de acceso remoto y local para algunas aplicaciones.

II. NUCLEO DE COMUNICACION

En los útimos años se han diseñado algunos modelos de comunicación para proveer interconectvidad a nivel de procesos en una red [ver Tan85]. Uno de los modelos más utilizados es el de paso de mensajes. Se basa en la existencia de procesos que administran recursos remotos (servidores) y procesos que solicitan estos recursos

(clientes). El desarrollo de una aplicación distribuída dentro del modelo anterior requiere de la incorporación de dos módulos: un módulo servidor en la máquina donde se encuentra el recurso, y un módulo cliente en la máquina desde donde se quiere hacer uso del recurso remoto.

Basado en el modelo anterior, se desarrolló un conjunto de primitivas de comunicación, conformando un núcleo de comunicación cuyo objetivo es proporcionar una plataforma base, independiente del sistema operativo del computador, para facilitar el desarrollo de aplicaciones en ambiente de red local.

El núcleo presupone la provisión de software estándar de comunicación tanto a bajo nivel en la red (por ejemplo, protocolos basados en IEEE 802.3) como en sus niveles intermedios (por ejemplo, protocolo TCP/IP). Esta suposición no disminuye la flexibilidad del sistema, por cuanto la disponibilidad de estas herramientas está ampliamente difundida en la mayoría de los equipos que coexisten en una red local en la actualidad (UNIX, VMS, MS-DOS, Macintosh, etc.).

TCP e IP son protocolos complejos que requieren de una cantidad considerable de procesamiento, y le imponen una alta carga al computador en términos de consumo de memoria, tiempo de procesamiento, y número de interrupciones. Por este motivo, para utilizar estos protocolos, especialmente desde un PC, sin recargar su CPU es conveniente usar una tarjeta de comunicación del tipo "inteligente". Esta tarjeta debe tener suficiente memoria para procesar el protocolo TCP/IP y manejar buffers para la comunicación con la red. De esta forma se alivia a la CPU de la alta carga que representa el procesamiento de comunicación, aumentando así la eficiencia global del sistema.

Por otro lado, la interfaz de comunicación se sustenta en el uso de **sockets** [Exc88] como conjunto de rutinas de comunicación entre procesos. Esta herramienta, inicialmente incorporada en sistemas UNIX, se provee actualmente en la mayoría de los computadores, incluyendo recientemente micro-computadores. El núcleo de comunicación desarrollado, sin embargo, presenta al programador un nivel mayor de abstracción que el proporcionado por estas rutinas de comunicación, facilitando el proceso de desarrollo de aplicaciones en sistemas distribuídos.

2.1 Interfaz de Comunicación

La interfaz de comunicación ha sido diseñada como un conjunto de primitivas de alto nivel. Estas primitivas están agrupadas en dos bibliotecas de rutinas en lenguaje C. Una de ellas incorpora las primitivas para desarrollar un programa cliente tanto en ambiente Unix como DOS. La otra incorpora las primitivas para el desarrollo de un programa servidor en máquinas multiusuario (p.ej.: Unix).

Las primitivas de comunicación para un **programa cliente** son las siguientes:

connect_to_server (char *service, *hostname)

Inicia una conexión con un servidor. Recibe como parámetros el nombre del servicio (service), y el nombre del host donde éste reside (hostname). Como resultado entrega un identificador numérico, de tipo integer, del socket (**sock**) sobre el cual se realiza la conexión y que corresponde a un subproceso de atención creado por el servidor.

disconnect (int *sock)

Esta función permite cerrar una conexión establecida con un servidor. Recibe como parámetro el identificador del socket (sock). Esta función no devuelve un resultado.

read_from_server (int sock, char *buffer, int buffersize)

Esta función es utilizada para leer información enviada por el servidor a través de la red. Recibe como parámetros el identificador del socket (sock), el buffer donde se almacenarán los datos leídos (buffer), y el tamaño del buffer (buffersize). Si la operación se ejecutó exitosamente, entrega como resultado la cantidad de caracteres recibidos, en caso contrario devuelve -1 y despliega un mensaje de error en la pantalla.

write_to_server(int sock, char *buffer, int n)

Esta función permite enviar datos al servidor. Recibe como parámetros el identificador del socket (sock), el buffer de datos a enviar (buffer), y el número de caracteres a enviar (n). En caso de éxito en el envío de los datos, esta función entrega el número de caracteres transmitidos por la red, en caso contrario devuelve -1 y despliega un mensaje de error en pantalla.

block_socket (int sock)

Esta función deja el socket en modo de lectura síncrona. Esto significa que la función read_from_server retorna el control al programa sólo cuando se reciben datos desde la red. Recibe como parámetro el identificador del socket (sock).

unblock_socket (int sock)

Esta función es la inversa de la anterior, es decir, deja el socket en modo de lectura asíncrona. Esto significa que la función read_from_server retorna el control al programa, el cual si no tiene datos que leer en ese instante puede continuar con otra operación y/o re-intentar una lectura posteriormente

Set_Confirm (int sock, flag).

Esta función sirve para establecer confirmaciones de mensajes. Recibe como parámetros, el identificador del socket (sock), y una valor entero (flag) que indica ya sea confirmar (flag=1) o no (flag=0) los mensajes desde y hacia el servidor. (por omisión flag=0).

Un programa tipo cliente debe, en primer lugar, conectarse con un servidor, indicándole el servicio requerido. Una vez establecida la conexión se procede al intercambio de información a través de la red, hasta que el cliente cierre la conexión. En la fig. 1 se muestra un esquema básico de un programa cliente.

```
#include "clien_lib.h"                          /* interfaz de comunicación */
main()
{
.....
sock = connect_to_server(service, hostname);  /* conexión con el servidor */
.....
while (...)                                     /* loop con servidor remoto */
      {
        ......
        read_from_server(sock,buffer,buffersize);
        ......
        write_to_server(sock,buffer,n);
        ......
      }
      disconnect(sock);
}
```

Figura 1. Esquema de construcción de un programa cliente

La biblioteca de primitivas disponibles para un programa servidor permite desarrollar un programa que corriendo en una máquina multi-tarea puede atender concurrentemente a varios clientes ubicados en distintos nodos de la red.

Las primitivas de comunicación para un **programa servidor** son las siguientes:

run_server (char *service, *<nomproc>)

Esta función inicializa un servidor y lo deja esperando por requerimientos desde los clientes. Al recibir una conexión se genera un subproceso que atiende al cliente respectivo, volviendo el servidor a esperar por un requerimiento de otro cliente. Recibe como parámetros el nombre del servicio a ofrecer (service), y el nombre de la rutina que lo provee (<nomproc>) la cual es invocada desde cada subproceso. Esta última es desarrollada por el programador y puede hacer uso de las primitivas del núcleo. La función run_server retorna un valor entero indicando el éxito o fracaso de la operación.

read_from_client (char *buffer, int buffersize)

Esta función es utilizada para recibir información enviada por el cliente a través de la red. Recibe como parámetros el buffer donde se almacenan los datos recibidos (buffer), y el tamaño del buffer (buffersize). Como cada subproceso atiende sólo a un cliente, no es necesario indicar la identificación del socket correspondiente. Si la operación se ejecutó exitosamente, entrega como resultado la cantidad de caracteres recibidos, en caso contrario devuelve -1 y despliega un mensaje de error.

write_to_client (char *buffer, int n)

Esta función envía datos al cliente. Recibe como parámetros el buffer con los datos a enviar (buffer), y el número de carácteres a transmitir (n). Si la operación se ejecutó exitosamente, entrega como resultado el número de carácteres enviados, en caso contrario devuelve -1 y despliega un mensaje de error.

block_socket

Esta función deja el socket de conexión con el cliente en modo de lectura síncrona y es similar a la rutina incluída para un programa cliente.

unblock_socket

Esta función deja el socket de conexión con el cliente, en modo de lectura asíncrona y es similar a la rutina incluída para un programa cliente.

<nomproc>

Esta función debe ser desarrollada por el programador, y contiene el código de la aplicación, la cual otorga un determinado servicio a los clientes de la red (el sistema servidor de procesos, que se describe en la sección 3, fue implementado bajo este esquema). Es una rutina que corre como un subproceso del servidor, y es activada una vez que un cliente ha solicitado una conexión con dicho servidor.

Un programa servidor consta de dos partes. La primera inicializa el proceso servidor el cual queda a la espera de los requerimientos de clientes, y la segunda corresponde a la función <nomproc> mencionada arriba, la que contiene el código de atención al cliente haciendo uso de las primitivas de comunicación. En la fig. 2 se muestra un modelo de un programa servidor.

```
#include  "serv_lib.h"                    /* interfaz de comunicación */
main()
{
      run_server(service,  ServProc);     /* inicializa el servidor  */
}

ServPro:                                  /* desarrollo del servicio */
{
      while (...)                         /* loop de atención al cliente  */
      {
          ........
          read_from_client(buffer,buffersize);
          ......
          write_to_client(buffer,n);
          .......
      }
      exit();                             /* finaliza el subproceso */
}
```

Figura 2 Esquema de desarrollo de un servicio

III.- SERVIDOR DE PROCESOS

Utilizando las rutinas de propósito general descritas en la sección anterior, se desarrolló un sistema de acceso a recursos remotos, disponibles en los distintos computadores de la red (compiladores, bases de datos, comandos del sistema operativo, etc).

Cada nodo del sistema incluye dos módulos básicos, el módulo **cliente** y el módulo **servidor de procesos**. El primero es un proceso que permite invocar cualquier aplicación remota. El segundo es un proceso que, a través de una puerta de comunicación fija, escucha por requerimientos de clientes que solicitan servicios del nodo. Esto evita tener un servidor activo en la red por cada tipo de recurso que pueda ser invocado remotamente (ver fig. 3).

El servidor de procesos, una vez que recibe un requerimiento de conexión, crea un subproceso para luego quedar en espera de otro cliente. Este nuevo subproceso recibe el nombre del servicio requerido, y lo ejecuta, comunicando la entrada (input) y la salida (output) de dicho servicio al proceso cliente en el nodo remoto.

El esquema de administración de recursos es distribuído. En cada nodo existen dos archivos que permiten a un cliente obtener información de los nodos y sus aplicaciones disponibles. El primer archivo contiene una lista de las aplicaciones que pueden ser invocadas en el sistema junto con el nodo donde residen. El segundo, contiene los nombres de los distintos nodos de la red y la puerta de acceso al servidor de cada nodo. De esta forma es posible expandir o restringir el sistema a nodos y aplicaciones específicos para distintas máquinas.

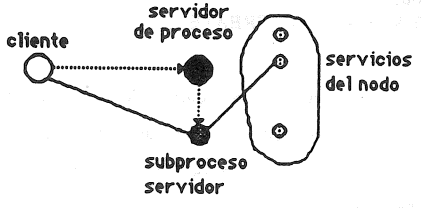


Figura 3: Esquema general del sistema.

Por sobre los protocolos básicos de la red (por ej.: IEEE802.3, TCP/IP) se incluye, opcionalmente, un protocolo del tipo **stop-wait**, útil en sistemas que requieran mayor confiabilidad (ver primitiva SetConfirm arriba). Este protocolo con una pérdida de eficiencia muy pequeña, permite asegurar la confiabilidad en los datos en un ambiente donde coexisten sistemas heterogéneos y no necesariamente confiables.

3.1 Utilización de Procesos

El módulo cliente para equipos Unix incluye dos procesos que manejan separadamente el *input* y *output* del terminal del usuario. Con esto se obtiene un mejor desempeño y una adecuada sincronización para ambas actividades. Los sistemas MS-DOS, al ser del tipo mono-tarea, requieren de adaptaciones en relación al número de procesos. En este caso un sólo proceso maneja el flujo de datos en ambos sentidos.

En el servidor de procesos, existe un solo proceso que interactúa con el cliente. De esta manera se evita una proliferación excesiva de procesos en el servidor ya que éste puede ser invocado concurrentemente por muchos clientes.

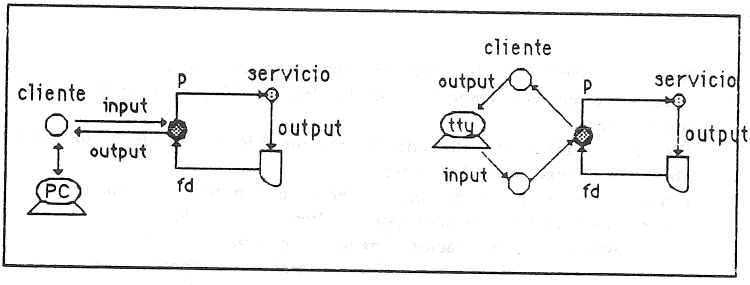
El mecanismo básico utilizado por el servidor de procesos, para otorgar acceso a aplicaciones remotas, es mediante la función **popen** que provee UNIX. Esta función permite abrir un "pipe" a un proceso para leer o escribir sobre él, realizar I/O redireccionando la salida de la aplicación, para luego escribir los comandos sobre ella en el identificador de archivo que retorna la función popen.

Luego, con las siguientes instrucciones se puede tener acceso a la aplicación en forma sencilla (en este ejemplo el servidor de aplicaciones invoca al intérprete FLISP):

```
p = popen ( "Flisp > output" , "w" );
fd = open ("output" , O_RDONLY | O_CREATE );
```

Luego, la aplicación remota (Flisp) recibe los datos del cliente mediante el identificador **p** del "pipe" y escribe sus resultados en un archivo temporal (output) donde el servidor (en este caso un subproceso del servidor de procesos) puede leerlos usando el identificador de archivo **fd**. Este procedimiento se puede apreciar en la fig. 4, tanto para equipos Unix como equipos MS-DOS.

Una vez que se obtiene acceso de I/O con la aplicación, es sencillo conectar este I/O con el cliente. Para esto se tiene una iteración indefinida en que todo el *output* de la aplicación se transmite al cliente por la red y todos los comandos que el cliente envía se escriben en el *input* de la aplicación.



a) sistema MS-DOS b) sistema Unix

Figura 4 Esquema del acceso a una aplicación remota

Esta solución es satisfactoria, ya que permite ejecutar aplicaciones remotas redireccionando el I/O de un proceso hacia el cliente. Esto se validó corriendo algunas aplicaciones como LISP, ORACLE, SCHEME, TELNET, y comandos Unix tales como who, ps, ls, etc. El sistema desarrollado se compone actualmente de servidores de procesos que corren en máquinas Unix y clientes que corren en equipos Unix y micro-computadores MS-DOS.

Los pasos que se verifican en el **proceso cliente**, cada vez que el usuario selecciona un servicio son los siguientes :

a) Conexión con el servidor.
b) Si el servidor responde, se le envía el nombre de la aplicación a ejecutar.
c) El cliente espera una señal del servidor que indica que éste posee recursos suficientes para atenderlo. De no ser así, se aborta la comunicación.
d) Se envían los archivos necesarios para ejecutar la aplicación remota (p.ej. un programa a compilar). Luego, el cliente itera hasta que ocurra uno de los eventos siguientes:
   - El servidor indica el término de la aplicación remota.
   - El cliente envía un comando de término de conexión
   - El usuario activa el menú principal (caso de MS_DOS).
   - El usuario activa otra ventana (caso de MS-DOS).
e) Finalmente se reciben los archivos modificados o generados durante la ejecución de la aplicación.

Con respecto al **servidor de procesos** de un nodo, éste está permanentemente escuchando en una puerta si existe un cliente que desea conectarse. Al recibir este requerimiento, el servidor procede a crear un subproceso que será el encargado de atender al cliente respectivo, mientras el servidor sigue esperando a otro cliente. Este nuevo proceso generado ejecuta la rutina **<nomproc>**, la que realiza las siguientes operaciones:

a) Recibe el nombre de la aplicación que se desea ejecutar.
b) Crea un directorio temporal y se localiza en él para ejecutar allí el resto de las operaciones.
c) Ejecuta la aplicación deseada redireccionando su salida a un archivo temporal, abriéndolo luego para leer desde él.
d) Envía al cliente una señal que le indica que puede ser atendido.
e) Recibe los archivos enviados por el cliente (si los hay).
f) Itera hasta que la aplicación finalice o el cliente le envíe una señal de término. En este proceso, el servidor lee el archivo temporal, output de la aplicación, enviándole al cliente toda la información que allí exista. Luego se leen los comandos que envía el cliente para comunicárselos a la aplicación.
g) Finalmente, al término del proceso se envían de vuelta los archivos modificados o creados durante la ejecución de la aplicación (si los hubo), y se procede a borrar el directorio temporal.

4 INTERFAZ DE USUARIO

La interfaz de usuario para acceder al sistema anterior fue desarrollada para dos ambientes, uno simple al estilo de comandos Unix y otro más sofisticado y "amistoso" orientado a micro-computadores.

4.1 Interfaz de comando

El sistema servidor de procesos está actualmente instalado en equipos Unix V y Unix 4.2 BSD, con diferencias mínimas en su implementación. Una interfaz simple de usuario disponible tanto en clientes de terminales ASCII como de PCs consiste de un comando que conserva la sintaxis típica de los comandos en Unix. En su expresión más simple, el comando es:

net <nombre-servicio>

Donde, <nombre-servicio> es el nombre del servicio remoto a ejecutar (p.ej: compilador, editor, comando). En su expresión más compleja, el comando es:

net [-c] [-n nodo] <nombre-servicio> [-f <lista de archivos>]

La opción "-n nodo" indica el lugar (nodo) de ejecución del servicio invocado. Esta opción es útil cuando un servicio reside en varios nodos. La opción -c se usa para aquellas aplicaciones, que poseen comandos de una sola tecla (por ejemplo editores como **emacs** o **vi**), que requieren que cada tecla presionada por el usuario sea comunicada inmediatamente a la aplicación.

La opción -f se usa para especificar el uso de archivos, indicando la separación entre la aplicación a usar y los archivos a transferir. Con esta opción se evita la complejidad de manejar la transferencia de archivos con posterioridad a la invocación de la aplicación, en aquellas aplicaciones que así lo hacen normalmente. Algunos ejemplos de este comando se muestran a continuación:

```
net -n juncal who              (consulta de usuarios activos en nodo juncal)
net -c emacs new.c             (edición de new.c en un editor emacs de la red)
net Lisp -f  x.lsp  y.lsp      (uso de intérprete Lisp de la red con dos archivos)
net sqlplus -f A.sql  B.sql    (uso de base datos sqlplus de la red, con dos archivos)
```

4.2 Interfaz para PCs

Una Interfaz de usuario basada en un sistema de ventanas permite hacer un uso más eficiente de la pantalla de un computador. Debido a la posibilidad de tener varias ventanas concurrentemente en la pantalla, cada una desplegando la salida de una aplicación remota distinta, es que este tipo de interfaz es muy adecuada en ambientes de red.

Mediante esta interfaz se puede tener acceso tanto a aplicaciones remotas como a dispositivos de gran capacidad, tales como discos, impresoras, servidores de comunicación, etc., transparente a su localización en la red y forma atractiva al usuario. Las aplicaciones que corren concurrentemente en las distintas máquinas de la red, despliegan sus resultados en una ventana independiente en la pantalla del PC.

Para construir la interfaz se desarrollaron diversas rutinas de manejo de ventanas, y menús, utilizando el lenguaje C y las rutinas gráficas de Quick-C. Mediante estas rutinas se provee actualmente un prototipo de interfaz de usuario que incorpora

diversas facilidades para el acceso a recursos remotos. Cada ventana abierta en la pantalla representa una conexión con un subproceso servidor de la red.

La interfaz provee un mecanismo para que el usuario pueda ir activando las ventanas abiertas. Esto le permite pasar de una aplicación remota a otra con sólo presionar una tecla. Con esta facilidad el usuario puede dejar, por ejemplo, corriendo una larga compilación en una ventana, imprimiendo un documento en otra, mientras hace consultas a una base de datos remota en la ventana activa. Mediante otra tecla, es posible dejar superpuestas las ventanas, de modo que la ventana activa sea la única que se muestra visible, y a un tamaño adecuado para la aplicación.

La fig. 5 presenta un esquema simplificado de esta interfaz, donde se muestra una sesión real de uso de recursos de la red: usando un intérprete de la red en la ventana activa, en otra realizando una conexión a un computador de la red, y en otra ventana invocando un comando remoto. El esquema de menús es jerárquico, en donde existe una subdivisión por categorías de recursos en la red (comandos, aplicaciones, lenguajes, etc). Por ejemplo, la categoría lenguajes de programación, se subdivide en lenguajes funcionales (p.ej.: FP, LISP, Scheme, etc), imperativos (p.ej.: Pascal), lógicos (p.ej.: Prolog), etc. En caso de existir una aplicación residente en más un nodo, la interfaz consulta al usuario por el nodo respectivo, y en caso de omisión el sistema escoge uno.

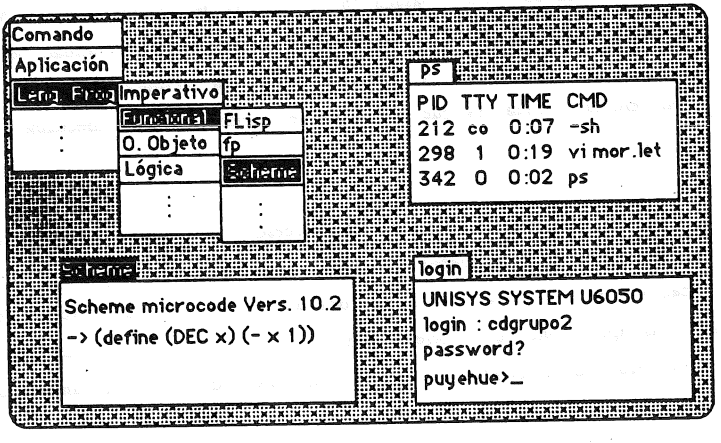


figura 5. Esquema de interfaz de usuario en PC

Actualmente, cada módulo cliente mantiene una pequeña base de datos con las aplicaciones de la red y los nodos donde residen, involucrando así una administración descentralizada de recursos. Se considera que ésta es adecuada para el tamaño actual de la red (no más de 10 nodos). En la medida que la red se expanda se pretende optar por un esquema administrativo que involucre la existencia de agrupaciones de nodos, por ejemplo de acuerdo a un patrón de actividad común de los nodos, asignándole a cada grupo un proceso localizador [Aur90], el cual queda residente en alguno de los nodos que conforman el grupo.

V ANALISIS DE DESEMPEÑO

El sistema desarrollado se enmarca dentro de un esquema de acceso del tipo cliente/servidor. Este es un modelo típico de interacción entre computadores conectados en red local, y es especialmente útil a estaciones de trabajo con pocos recursos, conectadas a la red. La tendencia, hoy en día, es hacia un aumento progresivo de equipamiento computacional conectado en redes locales, donde coexisten algunos equipos que actúan como servidores y especialmente equipos que son clientes de los primeros.

Por ende, es importante analizar el impacto que la red, a través de sus diversos componentes, representa en el desempeño de un sistema tipo cliente/servidor. Con esta motivación, se procedió a desarrollar un sistema simple de medición de desempeño, mediante la incorporación de rutinas, en el módulo cliente, que permitieran tanto la detección de tiempos reales de respuesta desde el servidor como la generación controlada de requerimientos desde el cliente.

En las mediciones se escogieron fundamentalmente aplicaciones remotas interactivas, de modo de medir el impacto de la red antre una consulta y la llegada de la respuesta correspondiente. Para ello, además de las mediciones de tiempo en el cliente, se procedió a medir paralelamente los tiempos de respuesta de la aplicación ante consultas locales, para tener una medida del comportamiento del sistema independiente de la aplicación siendo utilizada. Además, las mediciones se realizaron en períodos de baja carga (p.ej. durante la noche), y con varias repeticiones para aislar el impacto de carga externa al sistema.

Una de las aplicaciones escogidas fue el sistema administrador de base de datos SQLPLUS, localizado en un computador Unix con procesador Intel 80386, seleccionándose consultas típicas tales como Select, Join y Help, a una base de datos pequeña. Con el fin de analizar la degradación del tiempo de respuesta frente a un aumento de carga en el servidor, se procedió a generar, automaticamente, distintas cargas de trabajo (reales), desde un cliente, consistentes en usuarios invocando una determinada consulta a través la red.

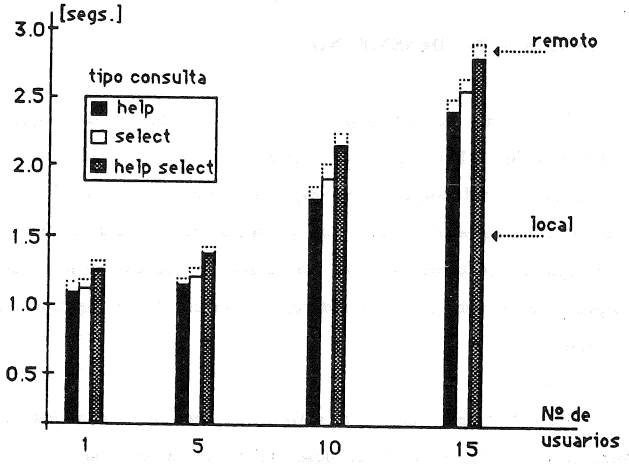


Fig. 6 Tiempos de respuesta en acceso a una base de datos

La fig. 6 muestra el tiempo de respuesta promedio a consultas tanto locales como remotas. Se puede apreciar que la degradación que produce el sistema de acceso remoto es relativamente pequeña. Por otro lado, aunque no presentado en la fig. 6, para una carga de más de 20 usuarios, se observó una degradación significativa en el tiempo de respuesta, pero ésta era debida exclusivamente a la saturación de la máquina que corría la aplicación.

Por otro lado, para el caso de acceso remoto se realizó un análisis de degradación porcentual en los tiempo de respuesta. El resultado de ese análisis se muestra en la fig. 7. Se aprecia que existe un mayor incremento en el tiempo de respuesta, al aumentar de aproximadamente 5 a 6 usuarios. Esto se podría explicar debido a que el software de la red habilita, actualmente, colas de atención simultánea

de hasta 5 clientes, debiendo los clientes en exceso de esta cantidad repetir la operación de acceso a la cola.

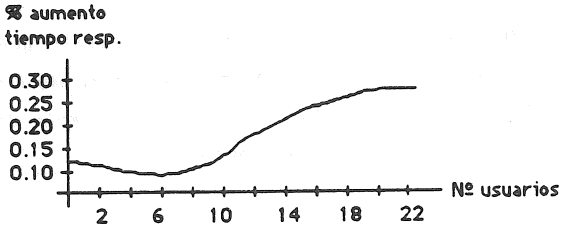


Figura 7. Variación porcentual en tiempo de respuesta remoto

Los resultados mostrados en esta sección, no pretenden mostrar un análisis exhaustivo de medición del desempeño del sistema. Su objetivo fue por un lado, entregar una visión general de su comportamiento frente a algunos requerimientos, y por otro construir un primer prototipo de evaluación de desempeño susceptible de mejorar a futuro.

Como próxima etapa, se proyecta incorporar herramientas de monitoreo en el núcleo de comunicación descrito en la sección 2, para adaptar el prototipo anterior a un sistema que permita desagregar el análisis del desempeño en distintos niveles de servicios de una red. Luego, se espera construir un monitor activo de sistemas distribuídos, que permita tanto medir como proyectar el desempeño de éstos mediante la incorporación de los datos de mediciones a un modelo de red de espera.

IV.- CONCLUSIONES

Mediante este trabajo se han construído herramientas computacionales que facilitaron el desarrollo de un sistema de acceso transparente a recursos distribuídos en una red local universitaria, incluyendo desde simples comandos hasta aplicaciones diversas. De paso, se han incorporado herramientas que permiten obtener el desempeño de este tipo de sistemas.

La interfaz de comunicación desarrollada permite ocultar los detalles de bajo nivel de la red, haciéndola transparente tanto al programador de sistemas distribuídos

como a los usuarios. Permite separar los aspectos de comunicación de la problemática inherente a la aplicación propiamente tal.

La interfaz de usuario disponible a nivel de PCs, permite que el usuario vea los recursos remotos como si residieran en su PC. Además, permite el uso concurrente de varias aplicaciones remotas. Esto se logra haciendo que cada aplicación despliegue su salida en una ventana independiente del micro-computador.

BIBLIOGRAFIA

[Aur90] Aurtenechea F, Wallem A (1990) Una Herramienta para el Desarrollo de Sistemas Distribuídos en una Red Heterogénea. **XVI Conferencia Latinoamericana de Informática**, Asunción, Paraguay, 1066-1080, Sept.

[Che88] Cheriton D. (1988) The V Distributed System. **Communicationof the ACM**, March, Vol 31, No. 3, 314-333.

[Exc88] Excelan Inc. (1988) Socket Library Application **Program Interface User's Guide.**

[Gla89] Glass L. (1989) "Building Heterogeneous Networks". **Byte,** (Sept. 1989).

[Iso81] Iso (1981) Open System Interconnection, Basic Reference Model, (ISOdraft proposal 7498) **ISO TC97/SC16. Computer Communications,** 15-65

[Met76] Metcalfe R., Boggs R. Ethernet: Distributed Packet Switching for Local Computer Networks. **Communicaction of teh ACM**, Vol. 19, Nº 7, 395-404.

[She86] Shelzer, A., Popek G.(1986) Internet Locus: Extending transparency to an Internet environment, **IEEE Transactions on software Engineering**, Vol SE-12, No 11, 1067-1075.

[Tan85] Tanenbaum, A. y Van Renesse, R. (1985) Distributed Operating Systems. **Computing Surveys**, Vol. 17, no. 4, 419-470

[Tan88] Tanenbaum, A. (1988) Computer Networks. Prentice Hall, New Jersey.