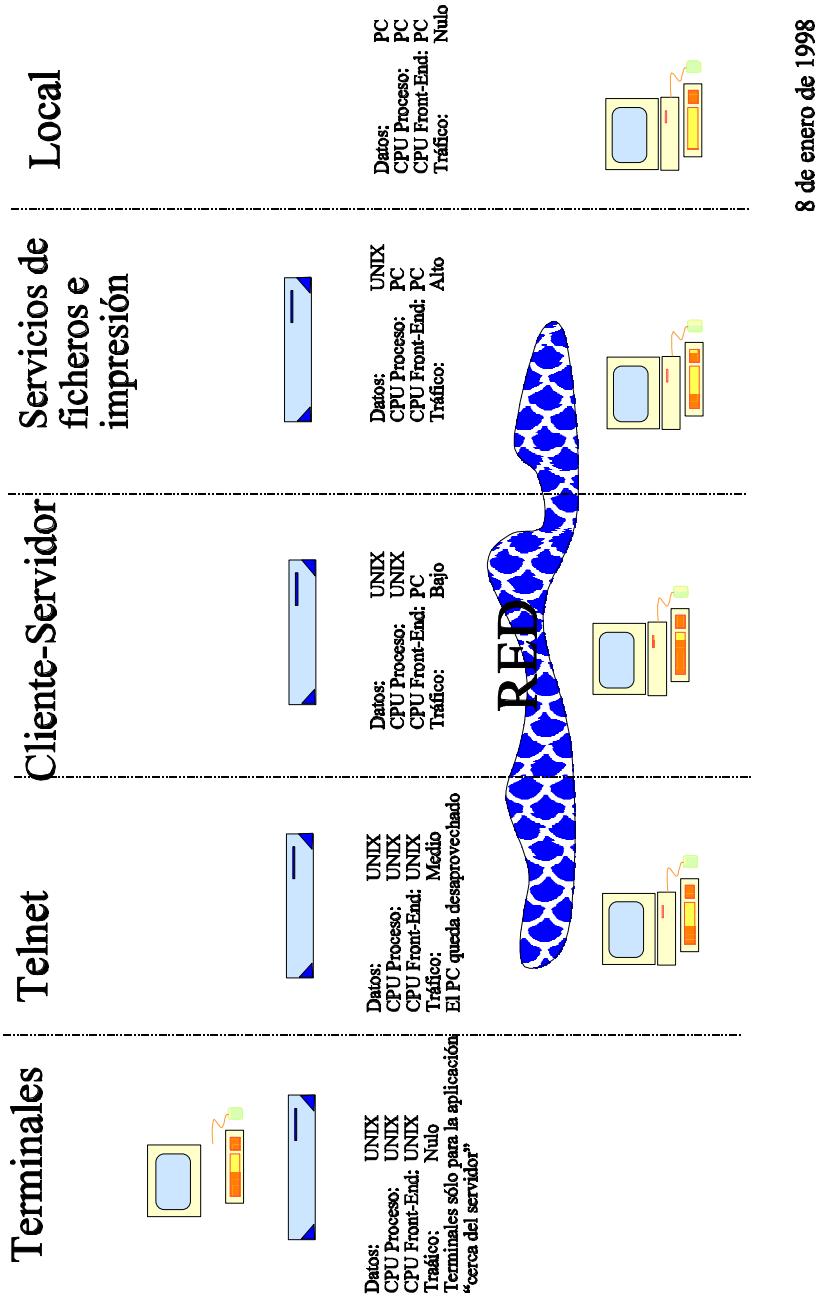
Redes de ordenadores

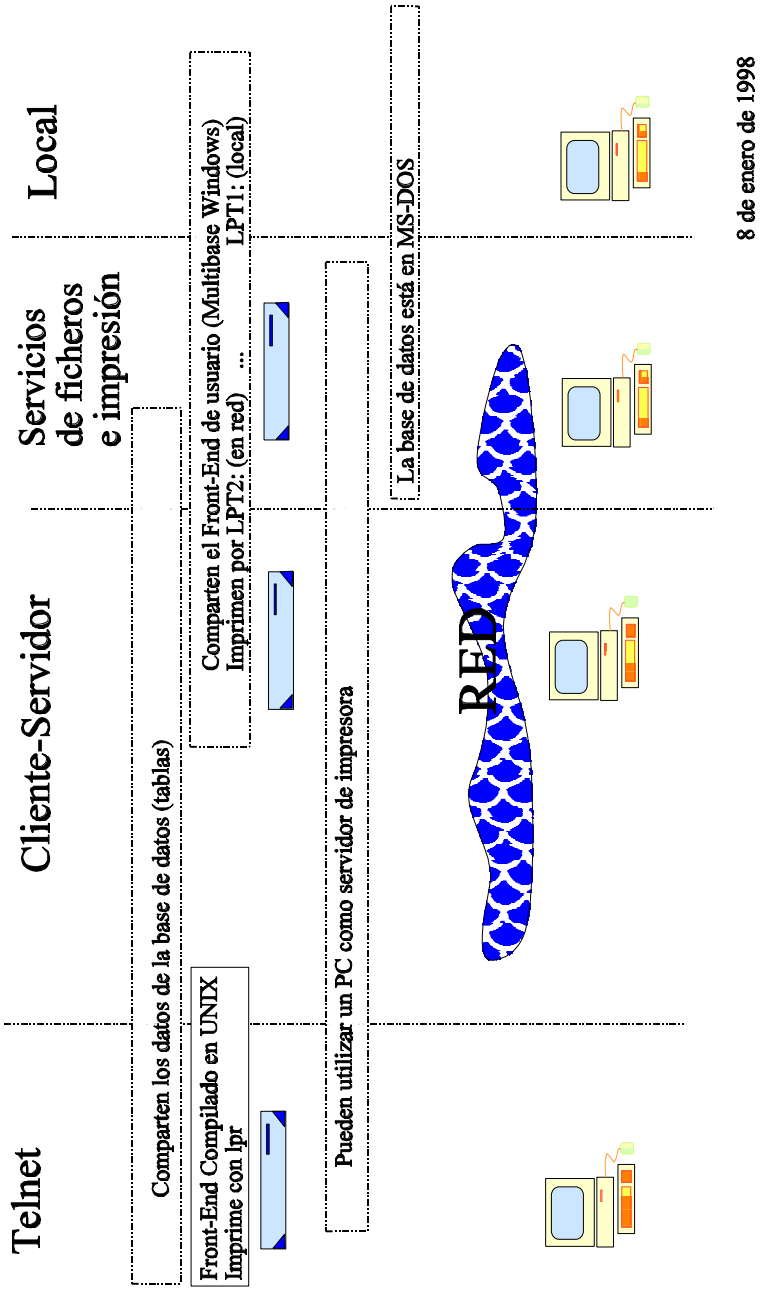
Cliente servidor, NFS y NIS

Grupo de sistemas y comunicaciones
Juan Jesús Muñoz Esteban
jjmunoz@gsyc.inf.uc3m.es

9.Modelo cliente/servidor



9.2 Funciones del cliente/servidor



9.3 Comunicación entre programas

Las aplicaciones no se construyen de forma monolítica. Inicialmente se utilizaban llamadas a subrutinas y saltos (goto). Pero en los años 70 Wirth, Dijkstra y muchos otros abogaron por la programación estructurada, que descompone los programas en procedimientos y funciones que se invocan entre si pasandose parámetros por valor o por referencia. En los años 80 se reorganiza el código agrupando estos procedimientos (ahora llamados métodos) con sus variables, constituyendo objetos.

La llamada a procedimiento es pues un concepto aceptado y extensamente utilizado por los programadores, que además por el volumen de las aplicaciones están acostumbrados a que estos pedazos de código estén en diversos ficheros, se compilen por separado y en la última fase, montado, se junten para producir un único ejecutable.

Además la programación concurrente enseña que el flujo de control no tiene por qué ser único, sino que un proceso puede albergar varios hilos paralelos.

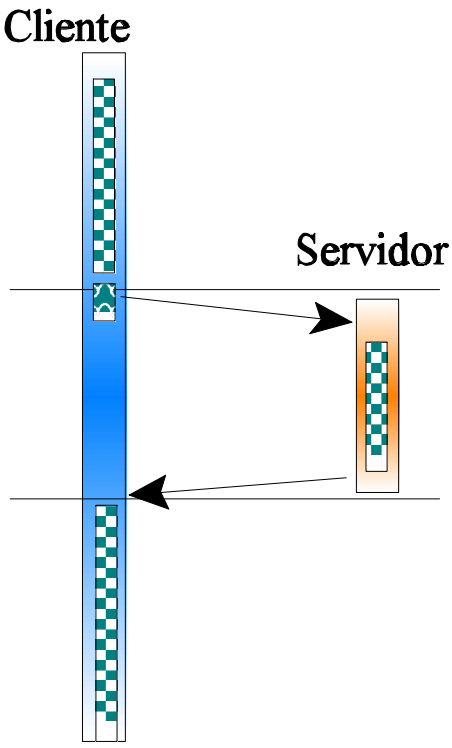
Los sistemas distribuidos incorporan como novedad que los distintos hilos se ejecuten no solo en pseudoparalelismo en una misma máquina, sino en ordenadores distintos que se intercomunican mediante una red.

Podríamos pensar que ésto supondrá hacer más compleja la aplicación, y que el programador necesitará nueva formación para poder aprovecharlo, pero no es así. Utilizando el concepto de invocación de función se tiene un nivel de abstracción suficiente para ocultar los detalles de dónde se ejecutará cada parte del código (una misma aplicación puede luego compilarse como cliente/servidor o como local) y dejar los detalles de la distribución a la generación del ejecutable y la configuración.

Para el programador es transparente. Por supuesto, quedan detalles como que un servidor puede recibir llamadas de varios clientes, o que las velocidades de ejecución relativa si reflejarán si funciona en local o en remoto, pero funcionamente nada cambia.

9.4 RPCs

Son un mecanismo genérico de petición de un servicio a otra máquina. Para el programador, es idéntico al de la llamada a un procedimiento local . La funcionalidad es similar a la del nivel de sesión OSI. Supone la transferencia de control entre partes de código que se ejecutan en máquinas diferentes.



El cliente al invocar una llamada remota, fuerza que se encapsulen los datos de los parámetros y viajen por la red hasta el ordenador destino, donde se crea un entorno para ejecutar la función servidora. Una vez ejecutada ésta, se encapsula el resultado para que el cliente, al recibirlo, pueda seguir con la ejecución.

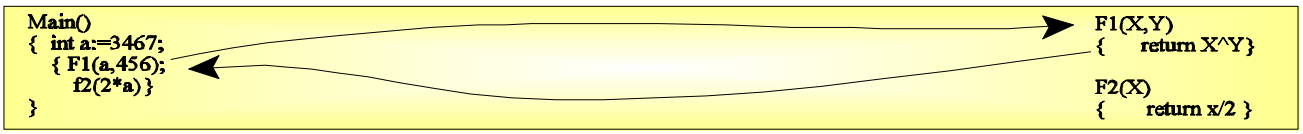
La invocación supone un tiempo empleado en viajar los parámetros y los resultados, pero puede que la ejecución de esa función en el servidor compense en cuanto a velocidad, además de que probablemente aporte otras ventajas (compartición de datos para otros clientes...)

El cliente puede, a diferencia de lo que ocurre en las funciones locales, seguir haciendo otras operaciones, y solo quedar bloqueado esperando la respuesta cuando los resultados le sean imprescindibles.

Este paso de mensajes también puede utilizarse para sincronizar procesos.

9.5 Implementación de RPCs

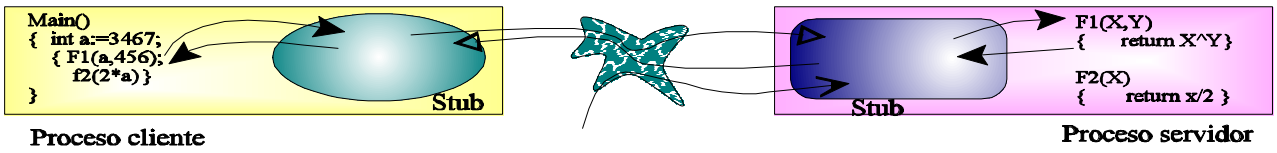
Las invocaciones a LPCs se suelen basar en el uso de una pila. El compilador ante una llamada del tipo Nombre_de_procedimiento(Parámetro1, Parámetro2), procede a meter en la pila los parámetros, y a continuación realizar un salto a posición del código donde se alberque la subrutina asociada al nombre del procedimiento.



Proceso cliente (y único)

En el caso de las RPCs este mecanismo debe sustituirse por el paso de mensajes, de manera que el los parámetros se codifiquen en un mensaje junto con la identificación del procedimiento, y se envíen (según los parámetros de configuración) al ordenador adecuado, que debe estar preparado para aceptar la llegada del mensaje, interpretarlo inequívocamente, ejecutar el procedimiento solicitado y generar un mensaje de respuesta.

El cliente tras la invocación deberá quedar bloqueado pero preparado para recibir un mensaje de la red, extraer de él el resultado y devolver el control al hilo de la aplicación que realizó la invocación con el valor cargado como lo habría estado en una LPC.



Proceso cliente **Proceso servidor**

! El stub o intermediario es el código (generado automáticamente por el sistema de RPCs utilizado) que suplanta en el clinente la posición de la función invoca y envía por la red los parámetros, queda a la espera del resultado lo devuelve como si hubiese ejecutado él mismo.
! En el servidor el stub es el programa principal, consistente en un bucle que espera que le llegue por la red la petición de una de las funciones preparadas, extrae los parámetros, invoca la verdadera función y envía un mensaje de respuesta resultado.

9.6 Semántica de las RPCs

Una importante diferencia con las LPCs es que cuando se invoca una operación local, o se obtiene un resultado, o el fallo también provoca la destrucción del cliente. Una RPC puede no tener resultado(el servidor puede caer sin afectar al cliente), y en ese caso hay que plantearse qué hacer (reintentarlo, probar con otro servidor, desistir).

La semántica de la invocación puede ser:

Exactly once :Es lo que desearíamos para mantener una absoluta transparencia, pero no es trivial de conseguir (hay que poner muchas comprobaciones...)
At most once: ejecutalo si puedes, pero nunca más de una vez (no reiterar)
At least once: Reintenta un número (razonable) de veces, hasta conseguirlo.

La invocación reiterada, provocada por la no recepción del resultado en un plazo razonable, puede conducir a fallos. Puede que la ejecución en el servidor culminase correctamente y la pérdida del mensaje con el resultado provoque esa nueva invocación, que puede alterar el resultado pretendido (ejemplo: orden de retirada de efectivo). La anomalía es consecuencia directa de repetir la operación.

No es lo mismo que el hecho de que el resultado de dos invocaciones idénticas pueda ser distinto porque se ejecutan en momentos distintos.

Las operaciones que pueden invocarse repetidas veces sin que por ello se altere el efecto que producen se llaman *idempotentes*.

Ejemplos:

Consultar el saldo: IDEMPOTENTE (aunque en ese momento otro te ingrese)
Descontar dinero de una cuenta: NO IDEMPOTENTE
Consultar los últimos movimientos: IDEMPOTENTE
Realizar una transferencia: NO IDEMPOTENTE

9.7 Problemática de las RPCs

- El servidor se ejecuta en otro entorno
- No hay variables globales compartidas
- El paso de parámetros es por mensajes, no mediante pila
- Hay que tener en cuenta las respectivas representaciones de los tipos de datos: (enteros de 4 o más bytes, coma flotante con más o menos bits...)
- Los procesos se ejecutan en máquinas distintas: ¿usan ASCII o EBCDIC? ¿ambas lo mismo?
- Los procesadores tratan de forma distinta los bits: big endian (68000, SPARC) versus little endian (Intel, VAX), distintos alineamientos de los registros...
- no se pueden pasar punteros (una dirección en un espacio de memoria no tiene sentido en otra máqina): Hay que pasar los parámetros por copia (valor, no por referencia)
- ¿Cómo pasar ADTs (pilas, colas) o tipos complejos (arrays, estructuras)?
- El proceso invocante debe bloquearse esperando un valor, pero su stub también, con capacidad para recibir un mensaje por la red.
- Problemas de semántica (transparencia anterior)
- ¿Manejo de excepciones?
- Rendimiento: un mensaje tarda en una LAN del orden de milisegundos. Puede resultar más lento el trasiego que lo que llevaría su ejecución local (si el cliente dispone de los datos necesarios para ejecutarla por sí mismo).

Como realizar traducciones de cualquier sistema con cualquier otro sería muy complejo, cada sistema define un formato de representación para la codificación en línea, de manera que al enviar los datos por la red realiza un proceso denominado marshalling que representa los valores a intercambiar (de tipos predefinidos) dentro de los mensajes que van por la red (nivel de presentación).

Los sistemas más conocidos son ASN.1 de OSI, y XDR de ONC (SUN).

La similitud entre el formato de red y el del procesador influye en las prestaciones.

9.8 SUN RPC

Hay varias especificaciones de RPCs: Courier, de Xerox (la primera), CEDAR (asociada al lenguaje MESA de Xerox), ARGUS del MIT, DCE, RMI de JAVA..

Una muy difundida (liberada desde el principio en una RFC) es ONC, de Sun, usada en NFS y NIS. Existen realizaciones de libre distribución, incluyendo herramientas de ayuda al programador.

Se apoya en XDR (External Data Representation) para obtener la funcionalidad de nivel de Presentación: Resuelve problemas con orden de bytes en enteros y números de coma flotante, cadenas, etc., proporcionando un formato canónico de transmisión y rutinas de transformación entre él y cada formato local.

F1(X: IEEE_Float; Y: Integer)
F2(Y: CHAR)

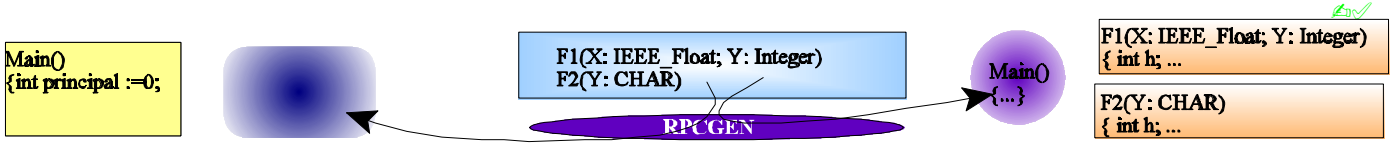
Ofrece un interfaz de alto nivel, sencillo, para definir el API entre clientes y servidores: nombres de funciones y sus parámetros, declarados según los tipos predefinidos. También hay un interfaz de bajo nivel para cambiar el nivel de transporte, ajustar temporizaciones y número de retransmisiones, autenticación, etc.

A partir de esta definición, la herramienta *rpcgen* genera dos ficheros con código:

- el stub cliente (para enlazar tras compilar), con una implementación de todas las funciones definidas anteriormente pero con código para generar mensajes por la red
- el stub servidor, con un procedimiento main() que registra las funciones en el portmapper y entra en un bucle sin fin que espera que le llegue un mensaje, invoca la función pedida en el mismo y genera un mensaje con el resultado.

Los distintos equipos de programadores deben escribir:

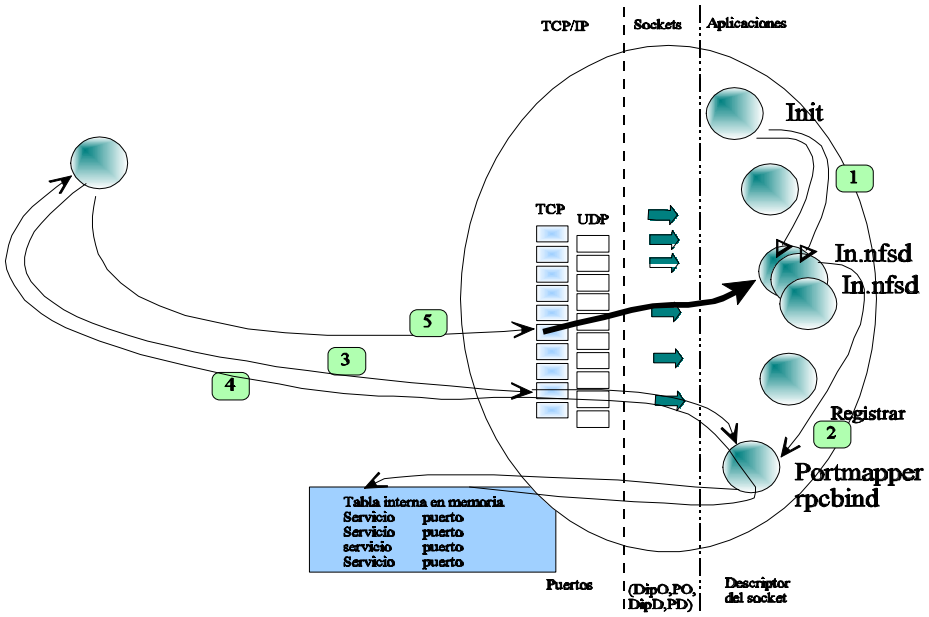
- El programa principal del cliente, pudiendo invocar las funciones ya definidas.
- El código de cada una de las funciones, que deberán montarse con el main() generado por rpcgen para construir el ejecutable del servidor



9.9 Configuración de las RPCs

Las RPCs son mucho más cortas y frecuentes que las invocaciones de otros servicios como telnet o mail. No es práctico lanzar el servidor cada vez que llega una.

El mecanismo normal (inetd) se sustituye por un "portero", al que el cliente tiene que consultar antes de la primera invocación en qué puerto hay un servidor del tipo deseado. portmap (rpcbind) escucha en un puerto fijo(111UDP). Los distintos servidores de RPCs al arrancar se "registran", y el portero reparte la carga.



Cada servicio tiene varios procedimientos, y responder a varias versiones. Los servidores registran su número de programa (normalizado, fichero /etc/rpc), puerto y versión. Se puede consultar con rpcinfo.

```
rpcbind   100000  portmap sunrpc rpcbind
rusersd   100002  rusers
ypserv    100004  ypprog
ypbind    100007
rstatd    100001  rstat rup perfmeter
nfs       100003  nfsprog
mountd    100005  mount showmount
nlockmgr  100021
```

9.10 Sistemas de ficheros en red: NFS

NFS (Network File System) es sistema de ficheros en red montando sobre RPCs. Un servidor sin estado contesta a peticiones independientes sobre partes del fichero.Es el estándar de facto en UNIX, y está disponible para DOS(suelen ser sólo clientes) y/o Windows 9x/NT, VMS, etc. y maneja el fichero adaptándose al S.O.

Existe un primer procedimiento de montaje del disco (que no pertenece a NFS), que "mapea" un directorio del servidor en otro (o un disco, como en MS-DOS) del cliente. Cuando el cliente solicita información, envía el file-handler obtenido en el montaje más el path del fichero hasta obtener un identificador para el fichero deseado. A partir de ahí puede pedir la parte que desee a base de RPCs, indicando el tamaño y el punto inicial.

El cliente almacena localmente los ficheros que tiene abiertos y la posición en cada uno. Si necesita bloquearlos, usa otro procedimiento de red: lockd. El cliente por tanto se implementa en el núcleo sel S.O. y redirige las peticiones remotas mediante las RPCs, incluso por adelantado (cache nfsiod), y es quien lleva el estado.

Si el servidor (nfsd) cae y se levanta en poco tiempo, el cliente no nota diferencia respecto de un momento de mucha carga (como si fuera lento) ya que es un servicio sin estado (cada petición es tratada como atómica, independiente de sus predecesoras). Por eficiencia suele haber varios procesos servidores en cola cíclica (hacer multithreading), ya que no consumen CPU y permiten paralelizar los accesos al disco. El servidor se configura especificando qué directorios se ponen disponibles por la red y para qué clientes. Utiliza autenticación a base de número de usuario (UID).

El protocolo define 16 procedimientos para manejar directorios (readdir, mkdir y rmdir, create, remove y rename), ficheros (read, write, setattr y getattr) enlaces (link, symlink, readlink) y sistema de ficheros (lookup y statfs).

En cuando a idempotencia, puede ocurrir que una orden de acceder a un fichero llegue después de otra de eliminarlo. En este caso el servidor lleva un caché con las últimas peticiones para identificar repeticiones y evitar mensajes de error al cliente.

9.11 File Handlers

Son referencias a los ficheros, que solo tienen una interpretación por parte del servidor (para el cliente es un token con el que referenciar).

SERVIDOR		CLIENTE DOS		Cliente UNIX
Estructura de ficheros:				
/d/sd/osd/f1		C:\		/
/d/sd/osd/f2		A:		/var
/d/otrod/....				/home
share /d (exporta /d: devolverá a los clientes autorizados el FH=8a56 correspondiente a este directorio)				
		Fase previa: Montar parte de lo exportado (no todo /d)		
	?	**NET USE R: servidor:/d/sd**		**mount -F nfs /home servidor:/d/sd**
	!		Obtiene el FH=8a56	
			Estructura de ficheros:	
		R:\osd\f1		/home/osd/f1
		R:\osd\f2		/home/osd/f2
		dir R:		**ls /home**
			? Readdir (FH=8a56, cookie=0)	
OK, 1 entry (no more): osd	**!**			
		type R:\osd\f1		**cat /home/osd/f1**
			? lookup (FH=8a56, osd)	
FH=1234	**!**			
			? lookup (FH=1234, f1)	
FH=FFA1	**!**			
			? getattr (FH=FFA1)	
OK, rw-r--r-, 1300bytes	**!**			
			? read (FH=FFA1, 1000,0)	
los 1000 primeros bytes	**!**			
			? read (FH=FFA1, 300,1000)	
los 300 últimos bytes	**!**			

DOS/UNIX hacen open, reads y close, cada uno con su FILE* y sus peculiaridades
El cliente NFS parte el camino en componentes y resuelve cada uno por separado
Semántica local: es DOS/i-nodo (el cliente) quien sabe hasta dónde ha leído del fichero
Cada petición es autocontenida (realmente hay más parámetros: UID, GID: v-node)

9.12 Ejemplo de NFS

mount -F nfs servidor:/directorio_del_servidor /directorio/local

cliente -> servidor	PORTMAP C GETPORT prog=100005 (MOUNT) vers=2 proto=UDP
servidor -> cliente	PORTMAP R GETPORT port=32819
cliente -> servidor	MOUNT C Null
servidor -> cliente	MOUNT R Null
cliente -> servidor	MOUNT C Mount /directorio_del_servidor
servidor -> cliente	MOUNT R Mount OK FH=00FC
cliente -> servidor	PORTMAP C GETPORT prog=100021 (NLM) vers=1 proto=UDP
servidor -> cliente	PORTMAP R GETPORT port=32773
cliente -> servidor	NLM C GRANTED FH=0000 FH=00FC PID=0 Region=0:0
servidor -> cliente	NLM R GRANTED FH=0000 granted
cliente -> servidor	PORTMAP C GETPORT prog=100003 (NFS) vers=2 proto=UDP
servidor -> cliente	PORTMAP R GETPORT port=2049
cliente -> servidor	NFS C NULL
servidor -> cliente	NFS R NULL
cliente -> servidor	NFS C GETATTR FH=00FC
servidor -> cliente	NFS R GETATTR OK
cliente -> servidor	NFS C STATFS FH=00FC
servidor -> cliente	NFS R STATFS OK

ls /directorio/local

cliente -> servidor	NFS C READDIR FH=00FC Cookie=0 (Usa el FH de /directorio_del_s)
servidor -> cliente	NFS R READDIR OK 20 entries (No more)
cliente -> servidor	NFS C LOOKUP FH=00FC fichero1
servidor -> cliente	NFS R LOOKUP OK FH=0D11 (Nuevo FileHandler para fichero1)
cliente -> servidor	NFS C LOOKUP FH=00FC otro_fichero
servidor -> cliente	NFS R LOOKUP OK FH=DC3B (Nuevo FileHandler para otro_fichero)
cliente -> servidor	NFS C LOOKUP
servidor -> cliente	NFS R LOOKUP OK FH=6AB1
cliente -> servidor	NFS C LOOKUP FH=00FC un subdirectorio
servidor -> cliente	NFS R LOOKUP OK FH=022F
cliente -> servidor	NFS C LOOKUP FH=00FC otro
servidor -> cliente	NFS R LOOKUP OK FH=6040
cliente -> servidor	NFS C LOOKUP
servidor -> cliente	NFS R LOOKUP OK FH=2F34

9.13 Administración centralizada: NIS

NIS (Network Information System , inicialmente llamado yellow pages) intenta facilitar la administración en un sistema distribuido:

- Mantenimiento de copias consistentes de ficheros de configuración
- Transparencia frente a usuarios (p.ej., contraseña en máquinas distintas)
- Administración centralizada.

Los servidores mantienen copias de ficheros con cierto formato (MAPAS), que consisten en tablas de una única clave creadas a partir de los ficheros de configuración (/etc/hosts, services, passwd, etc) y que se construyen con un Makefile.

Los clientes debe configurarse para que consulten parámetros (dirección IP dado un nombre, contraseña cifrada de un usuario) de estos mapas mediante RPCs en lugar de ir a sus ficheros locales (aunque en ocasiones, como el /etc/passwd, el mapa no sustituye sino que complementa la información local).

Cada vez que se hacen los cambios en el servidor “maestro" se propagan a los servidores “esclavos” (tolerancia a fallos por replicación). Los clientes consultan cada vez que necesitan algo, utilizando el servidor que primero les contestó la primera vez.

Dominio: conjunto de máquinas con un mismo gestor (mismos “mapas”).

Una máquina puede ser sólo cliente, sólo servidor o cliente y servidor a la vez.

Los programas que usan NIS son los mismos (sin cambio alguno) que los que toman sus datos locales. Simplemente cuando se pide un valor al sistema operativo, si este tiene habilitado NIS, resuelve el valor mediante una RPC en lugar de mediante una consulta local.

9.14 Gestión de red: SNMP

La gestión de red es el conjunto de tareas que se realizan para garantizar el funcionamiento de los sistemas de comunicaciones. Entre ellas cabe destacar la gestión de fallos (monitorización, detección, aislamiento y corrección: reconfigurar o reemplazar), accounting (identificar costes, autorizaciones de uso), configuración (control recopilación de datos), prestaciones, seguridad...

Para controlar tanta información hacen falta herramientas informáticas, pero tener una para cada tipo de dispositivo (y vendedor) sería inmanejable. Por ello existen plataformas (HP Openview, IBM Netview, SunNet Manager) sobre las que se construyen aplicaciones a medida sobre un repositorio común de datos.

Para la obtención de los datos las estaciones de gestión deben comunicarse con los dispositivos, que deben tener un agente de gestión.

Simple Network Management Protocol (RFC 1157 (de1990) y 1155) es un conjunto de estándares que incluye un protocolo, y un especificación de la estructura de una base de datos y un conjunto de objetos.

Cada dispositivo mantiene un objeto que describe su estado. La colección de todos los posibles objetos de la red llama MIB (Management Information Base). Los objetos se agrupan en 10 categorías (en la MIB-II: System, IP, TCP, SNMP...). Las estructuras de los datos se describen en ASN.1 (Abstract Syntax Notation One, OSI), y los datos se transfieren según sus BER.

Las estaciones de control sondean a los equipos recopilando datos y les envían nuevos valores de los mismos. Los agentes se limitan a responder, aunque en ocasiones envían traps ante eventos significativos definidos en la MIB. Hay 7 mensajes SNMP (get-request, get-next-request, get-bulk-request, set-request, inform-request y snmpV2-trap).

SNMPv2 (RFCs 1441 a 1452, 1993) mejora la seguridad
También es interesante la monitorización remota (MIB RMON).

9.15 Redes de PCs

Aunque Novell afirmaba en 1996 tener el 62% del mercado, el impacto de Windows-9x/NT y el interés por Internet confirman la expansión de TPC/IP para los PCs, y las aplicaciones que corren sobre winsock son cada vez más. En Otoño de 1998 la base instalada de NT supera la de Novell Netware.

El concepto básico de las redes de PCs ha sido el redirector, un elemento software que envía por la red los datos que normalmente van al driver de disco o al puerto de la impresora.

Las tarjetas de red para PCs son baratas y se configuran utilizando uno de estos tres mecanismos (que encapsulan las particularidades del driver y permiten que varios protocolos compartan la tarjeta):

NDIS (3com y microsoft): Config.sys carga el driver de la tarjeta (.dos) y el comando netbind lee los parámetros de protocol.ini.

ODI (novell): ejecutas LSL y luego el driver de la tarjeta (.com). La configuración se define en net.cfg, donde indicas los protocolos.

Packet Drivers (disponibles en la universidad de clarkson o crynwr, y si no hubiese uno específico, existe odipkt para usar el de ODI, que siempre existe). El driver es un ejecutable y en línea de comando toma su IRQ e IO-address.

Una vez cargado el driver de la tarjeta (o dejando que Windows 95 lo haga él solito), el stack de TCP/IP debe configurarse (direcciones, servidores de autenticación/licencias/DHCP), así como los discos e impresoras de red...

9.16 Ventanas en red: X-Window

X-Window es un sistema de ventanas desarrollado en el MIT que permite que programas que se ejecutan en una máquina saquen su salida (gráfica) en la pantalla de otra, según el modelo cliente-servidor.

El servidor corre en la estación de trabajo o el X-terminal (los hay para windows) y los clientes en los ordenadores que calculan lo que hay que dibujar (e invocan ordenes de dibujo del tipo línea, caja, círculo, bit-map...).

El modelo puede llevar a equívocos, porque el servidor corre en el puesto de trabajo, pequeño ordenador con capacidades gráficas, y los clientes pueden ejecutarse en distintos ordenadores, que no necesitan tener dispositivos gráficos.

La apariencia la controla un cliente llamado Window Manager. El más conocido es Motif, pero existen muchos. X/Open ha especificado CDE (Common Desktop Environment) para uniformizar los entornos de sobremesa sobre X.

La librería X ofrece funciones gráficas, de manera que por la red viajan ordenes del tipo: dibuja una circunferencia de tal radio y tal color, en esta ventana. Así el ordenador central se descarga de un trabajo que hacen mejor los puestos de trabajo.

ÍNDICE

Modelo cliente/servidor 2
Funciones del cliente/servidor 3
Comunicación entre programas 4
RPCs 5
Implementación de RPCs 6
Semántica de las RPCs 7
Problemática de las RPCs 8
SUN RPC 9
Configuración de las RPCs 10
Sistemas de ficheros en red: NFS 11
File Handlers 12
Ejemplo de NFS 13
Administración centralizada: NIS 14
Gestión de red: SNMP 15
Ventanas en red: X-Window 17