Índice

Capítulo 1. Introducción a los sistemas distribuidos

Objetivo: Que el alumno comprenda la importancia de los sistemas distribuidos en la sociedad actual, sus aspectos de diseño y sus principales retos tecnológicos.

1.1 Introducción

En años recientes, el avance en las tecnologías de cómputo y las telecomunicaciones han permitido una gran expansión de los sistemas de información, así como su alta disponibilidad, independientemente de su campo de aplicación. Las telecomunicaciones permiten la conectividad de un gran número de usuarios ubicados en cualquier parte del mundo por medio de la transmisión de voz, datos o video a través de una gran variedad de dispositivos. Diferentes redes de comunicación de área local (LAN), metropolitanas (MAN), así como de área amplia (WAN), pueden ser accedidas a través de Internet. Esto ha permitido que paralelamente surjan instalaciones de cómputo donde pueden ser desplegadas aplicaciones para realizar procesamiento distribuido de tareas. Estas nuevas facilidades ofrecen a los usuarios y organizaciones una gran flexibilidad para estructurar sus propios sistemas de información de una manera eficiente, así como la oportunidad de interactuar con otros sistemas de información de una manera distribuida. Como consecuencia, esto ha generado una gran dependencia de estos sistemas distribuidos para poder transmitir o procesar información. Tanenbaum [1996] define un sistema distribuido como una colección de computadoras independientes que aparecen ante los usuarios del sistema como una única computadora. El advenimiento de los sistemas distribuidos ha estado soportado en dos importantes innovaciones tecnológicas:

• El microprocesador.
• Las redes de área local.

1.2 Ventajas y desventajas de los sistemas distribuidos

1.2.1 Ventajas de los sistemas distribuidos con respecto a los sistemas centralizados

Entre las principales ventajas de los sistemas distribuidos con respecto a las computadoras centralizadas se encuentran:

• Economía: Los microprocesadores ofrecen una mejor relación precio/rendimiento que las computadoras centrales.
• Velocidad: Un sistema distribuido puede tener mayor poder de cómputo que una computadora centralizada individual.

• Distribución inherente: Implica que un sistema distribuido puede emplear aplicaciones instaladas en computadoras remotas.
• Confiabilidad: El sistema es consistente, aun si una computadora del sistema deja de funcionar.
• Crecimiento proporcional: Cada vez que se requiera mayor poder de cómputo en el sistema, solo se pueden adicionar los incrementos de cómputo requeridos.

1.2.2 Ventajas de los sistemas distribuidos con respecto a las computadoras aisladas

Con respecto a las computadoras aisladas, es decir, aquellas que no se encuentran conectadas a una red, los sistemas distribuidos tienen las siguientes ventajas [Coulouris, Dollimore & Kinderberg, 2001]:

• Datos compartidos: Permite que distintos usuarios tengan acceso a una base de datos o archivo común.
• Dispositivos compartidos: Permite compartir un recurso costoso entre distintos usuarios, como plotters o impresoras láser.
• Comunicación: Brinda la posibilidad de comunicación de usuario a usuario (telnet, correo electrónico, etc.).
• Confiabilidad: Facilita la repartición de la carga de trabajo entre las distintas computadoras con base en su funciones y capacidades, brindando una mayor flexibilidad y confiabilidad al sistema.

1.2.3 Desventajas de los sistemas distribuidos

A pesar de los diferentes beneficios que introducen los sistemas distribuidos, todavía existen diferentes retos que deben ser resueltos como los siguientes [Coulouris et al., 2001]:

• Software: Gran parte del software para sistemas distribuidos está aún en desarrollo.
• Redes: Los problemas de transmisión en las redes de comunicación todavía son frecuentes en la transferencia de grandes volúmenes de datos (por ejemplo, multimedia).
• Seguridad: Se necesitan mejores esquemas de protección para mejorar el acceso a información confidencial o secreta.
• Tolerancia a fallas: Las fallas operativas y de componentes aún son frecuentes.

1.3 Formas distintas de organizar n computadoras

La organización de cierta cantidad de computadoras se puede realizar usando alguno de los casos de los siguientes sistemas operativos:
• Sistema operativo de red.
• Sistema operativo distribuido.

• Sistema operativo de multiprocesamiento.

1.4 Aspectos del diseño de sistemas distribuidos

Transparencia
Es una característica de los sistemas distribuidos para ocultar al usuario la manera en que el sistema funciona o está construido, de tal forma que el usuario tenga la sensación de que todo el sistema está trabajando en una sola máquina local. Entre las principales transparencias deseables en un sistema distribuido están [Coulouris et al., 2001]:

• De localización: Los usuarios no pueden saber dónde se encuentran los datos localizados.
• De migración: Los recursos se pueden mover a voluntad sin cambiar su nombre.
• De réplica: Los usuarios no pueden ver el número de copias existentes.
• De concurrencia: Varios usuarios pueden compartir recursos de manera automática.
• De paralelismo: La actividad o consulta puede requerir procesamiento paralelo sin que el usuario lo perciba.
• De fallas: Cuando una computadora del sistema falla, esta es imperceptible para el usuario.
• De desempeño: El funcionamiento y velocidad de las máquinas donde se consulta es imperceptible para el usuario.
• De escalabilidad: El usuario ignora cuándo en el sistema se agrega otra computadora.

Flexibilidad
Facilita modificaciones al diseño inicial.

Confiabilidad
Permite que, en caso de que una computadora falle, otra la pueda sustituir en la realización de sus tareas asignadas.

Desempeño
Está en referencia a los tiempos de respuesta de una aplicación.

Escalabilidad
Permite que a la arquitectura actual se le pueda adicionar más poder de cómputo.

Repartición de la carga
Se debe analizar con qué equipos cuenta el sistema y los diferentes recursos de cómputo en cada uno de ellos, como capacidad de disco, velocidad de la red, etc. Los tipos de arquitectura a usar pueden ser:
• Servidores-estación de trabajo.
• Pila de procesadores.
• Multiprocesadores con memoria compartida.
• Multiprocesadores con memoria distribuida.

Mantenimiento de consistencia

Verificar que todos los conceptos involucrados con el sistema operativo, al operar en un esquema distribuido, sigan realizándose de manera correcta. Entre los puntos a observar están los siguientes:

• Modificación.
• Caché.
• Falla.
• Replicación.
• Interfaz de usuario.
• Reloj.

Funcionalidad
Implica que el sistema distribuido a implementar funcione de acuerdo con las metas trazadas y que permita hacer más eficiente el trabajo que antes se hacía usando un sistema centralizado.

Seguridad
Es importante considerar todos los factores de riesgo a que se expone la información en un ambiente distribuido, por ello se deben de implementar los mecanismos de seguridad que permitan proteger esta información.

1.5 Taxonomía de los sistemas distribuidos

Con base en su taxonomía, los sistemas distribuidos pueden clasificarse de la siguiente manera:
1. Sistemas con software débilmente acoplado en hardware débilmente acoplado. Ejemplo: Sistema operativo de red, como es el caso de NFS (Network File System - Sistema de archivo de red).

2. Sistemas con software fuertemente acoplado en hardware fuertemente acoplado. Ejemplo: Sistemas operativos de multiprocesador (sistemas paralelos).

3. Sistemas con software fuertemente acoplado en hardware débilmente acoplado. Ejemplo: Sistemas realmente distribuidos (imagen de sistema único).

Un caso de los sistemas distribuidos con software y hardware débilmente acoplado son los sistemas operativos de red. Algunas prestaciones de estos sistemas son:
• Conexión remota con otras computadoras.
• Copia remota de archivos de una máquina a otra.
• Sistema de archivos global compartidos.

Ejercicios

1. Menciona tres ventajas y tres desventajas de los sistemas distribuidos con respecto a los centralizados.
Respuesta:
Ventajas de los sistemas distribuidos con respecto a los centralizados:

- Economía: Los microprocesadores ofrecen una mejor relación precio/rendimiento que las computadoras centrales.
- Velocidad: Un sistema distribuido puede tener mayor poder de cómputo que una computadora centralizada individual.
- Distribución inherente: Permite emplear aplicaciones instaladas en computadoras remotas.

Desventajas de los sistemas distribuidos con respecto a los centralizados:

- Software en desarrollo: Gran parte del software para sistemas distribuidos aún está en desarrollo.
- Problemas de transmisión en redes: Los problemas de transmisión en las redes de comunicación pueden ser frecuentes.
- Seguridad: Se necesitan mejores esquemas de protección para mejorar el acceso a información confidencial o secreta.

2. Indica la importancia de la transparencia en los sistemas distribuidos.
Respuesta:
La transparencia en los sistemas distribuidos es importante porque permite ocultar al usuario la complejidad del sistema, haciendo que parezca como si todas las operaciones se realizaran en una única máquina local. Esto mejora la usabilidad y la experiencia del usuario.
3. Explica en qué consiste la transparencia de red en los sistemas distribuidos.
Respuesta:
La transparencia de red en los sistemas distribuidos se refiere a ocultar al usuario la ubicación de los recursos de red, como los datos o los servicios, para que puedan acceder a ellos de manera transparente, sin necesidad de conocer su ubicación física en la red.

4. Indica cuál es la diferencia entre sistemas fuertemente acoplados y sistemas débilmente acoplados.
Respuesta:
Los sistemas fuertemente acoplados tienen componentes que están estrechamente interconectados y dependen fuertemente unos de otros, mientras que los sistemas débilmente acoplados tienen componentes que pueden funcionar de manera más independiente y tienen menos dependencias entre sí.

5. Indica la diferencia entre un sistema operativo de red y un sistema operativo distribuido.
Respuesta:
Un sistema operativo de red permite la conexión remota con otras computadoras, la copia remota de archivos y el uso de sistemas de archivos globales compartidos, mientras que un sistema operativo distribuido es más complejo y permite la coordinación y el control de recursos distribuidos de manera transparente para el usuario.

6. Indica la diferencia entre una pila de procesadores y un sistema distribuido.
Respuesta:

Una pila de procesadores es un sistema en el que varios procesadores están apilados y trabajan en conjunto para realizar tareas, mientras que un sistema distribuido implica varias computadoras conectadas entre sí y trabajando juntas para lograr un objetivo común.

7. ¿Qué significa “imagen único” sistema en los sistemas distribuidos?
Respuesta:
"Imagen único" sistema en los sistemas distribuidos se refiere a la percepción del usuario de que está interactuando con un solo sistema, aunque en realidad hay múltiples sistemas trabajando juntos para brindar los servicios requeridos.

8. Indica cinco tipos de recursos en hardware y software que pueden compartirse de manera útil.
Respuesta:
Cinco tipos de recursos en hardware y software que pueden compartirse de manera útil son: datos, dispositivos de almacenamiento, dispositivos de entrada/salida, aplicaciones y servicios de red.

9. ¿Por qué es importante el balanceo de carga en los sistemas distribuidos?
Respuesta:
El balanceo de carga es importante en los sistemas distribuidos para garantizar que todos los recursos se utilicen de manera eficiente y equitativa, evitando la sobrecarga en algunas partes del sistema y garantizando un rendimiento óptimo en general.

10. ¿Cuándo se dice que un sistema distribuido es escalable?
Respuesta:
Un sistema distribuido se considera escalable cuando puede adaptarse fácilmente a cambios en la carga de trabajo o en el número de usuarios sin comprometer su rendimiento o funcionalidad.

11. ¿Por qué existe más riesgo a la seguridad en un sistema distribuido que en un sistema centralizado?
Respuesta:
Existe más riesgo a la seguridad en un sistema distribuido que en un sistema centralizado debido a la mayor complejidad y la necesidad de proteger la comunicación entre múltiples dispositivos y la información que se transmite a través de redes potencialmente inseguras. Además, la presencia de múltiples puntos de acceso puede aumentar las vulnerabilidades a ataques.

Capítulo 2. Redes de computadoras

Objetivo: Que el alumno revise los componentes básicos de las redes de comunicación para entender su importancia vital como infraestructura para la construcción de los sistemas distribuidos.

2.1 Introducción

Las redes de computadoras son un componente importante de los sistemas distribuidos. Una red de computadoras es una colección interconectada de computadoras autónomas que son capaces de intercambiar información [Tanenbaum, 1997]. El objetivo principal de las redes de computadoras es compartir recursos, de tal manera que todos los programas, equipos y datos se encuentren disponibles para quien lo solicite sin importar su ubicación. El uso de las redes de cómputo se ha incrementado durante los últimos años. La comunicación por computadora se ha convertido en una parte esencial de la infraestructura actual. La conectividad se usa en muchos aspectos y, con el crecimiento continuo de la Internet, las demandas de enlaces de mayor capacidad también han aumentado. En una red de cómputo, los datos son transmitidos entre computadoras usando secuencia de bits para representar códigos. La capacidad de transmisión de los datos, referida comúnmente como ancho de banda, es descrita en bits por segundo (bit/s). Las capacidades típicas [Black, 1993] de transmisión de datos se muestran en la tabla 2.1.

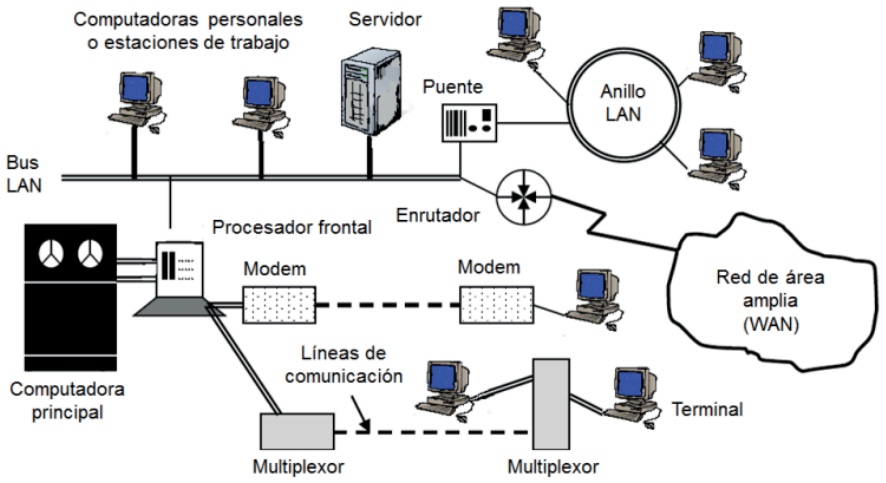
Tabla 2.1. Velocidad de conexión y usos en las redes de computadoras [Black, 1993]

Velocidad típica (bit/s) | Uso típico
0-600 | Telégrafo, terminales viejas, telemetría
600-2,400 | Terminales operadas humanamente, computadoras personales
2,400-19,200 | Aplicaciones que requieren respuesta rápida y/o rendimiento similar como transferencia de archivos
32,000-64,000 | Voz digital, aplicaciones de alta velocidad, algo de video
64,000-1,544,000 | Muy alta velocidad para múltiples usuarios, tráfico de computadora a computadora, conexión principal de red, video
Mayores a 1.5 MB | Conexión principal de red, video de alta calidad, voz digital múltiple

2.2 Principales componentes de una red de cómputo

Las redes de computadoras están integradas por diversos componentes, algunos de los cuales se muestran en la figura 2.1 [Black, 1993].

Figura 2.1. Componentes de redes de cómputo



Los principales componentes de una red de cómputo son [Black, 1993]:

• Medios de comunicación: Cualquier cosa usada para transportar datos en la forma de señales eléctricas (líneas telefónicas, líneas dedicadas o canal LAN).
• Macrocomputadoras: Computadoras donde residen las grandes bases de datos o información de una empresa bajo un ambiente centralizado.
• Terminales de cómputo: Dispositivos de entrada/salida de una computadora principal, puede consistir de teclado, lector óptico o cámara de vídeo para la entrada y monitor de video o impresora como salida.
• Enrutadores: Dispositivo que en la red de comunicación examina las direcciones de la red dentro de un protocolo dado y encamina los paquetes al destino por la ruta más eficiente previamente determinada.
• Modem: Dispositivo usado para convertir datos digitales seriales de una terminal transmisor en una adecuada señal analógica para un canal telefónico, así como para reconvertir la señal analógica a digital serial para que sea usada por una terminal receptora.
• Multiplexores: Estos dispositivos permiten que más de una terminal comparta la línea de comunicación y pueda resultar en una ventaja sustancial, al reducir el número de líneas usadas.
• Estaciones de trabajo: En un entorno de red es una computadora para un único usuario, de alto rendimiento, que sirve para diseño, ingeniería o aplicaciones científicas.
• Conmutadores: Dispositivo mecánico o electrónico que sirve para comandar el flujo de señales eléctricas u óptica.
• Procesador central: Usado todavía por algunos sistemas de cómputo, su función es manejar el procesamiento de comunicaciones en un entorno de macrocomputadoras, conectando por un lado los canales de comunicación y por el otro la macrocomputadora.
• Servidores: Su propósito es proporcionar rendimiento a estaciones de trabajo o computadoras personales o para proporcionar base de datos y servicios de impresión o correo en una red.

2.3 Modo de operación y conmutación

El modo de operación de un enlace de comunicación es una característica a considerar desde la perspectiva de un sistema distribuido. Usualmente para transmitir datos por un enlace de comunicación se ocupan tres modos:
1. Comunicación simplex: Este modo se presenta cuando los datos viajan en una sola dirección.
2. Comunicación half-duplex: Este modo permite que los datos viajen en dos direcciones, una a la vez.
3. Comunicación full-duplex: En este modo los datos viajan simultáneamente en ambas direcciones.
Para transferir datos, las redes utilizan comunicación conmutada, lo que permite a los dispositivos compartir líneas físicas de comunicación, los métodos [Tanenbaum, 1997] más usuales de comunicación conmutada son:

• Conmutación de circuitos: Crea una ruta única e ininterrumpida entre dos dispositivos que quieren comunicarse así que, mientras estos se comunican, ningún otro puede ocupar esa ruta.
• Conmutación de mensajes: En esta conmutación no existe un establecimiento anticipado de la ruta entre el que envía y quien recibe. Cuando el que envía tiene listo un bloque de datos, esta se almacena en la primera central de conmutación, para expedirse después como un salto a la vez. Cada bloque se recibe completo, se revisa y se retransmite, sin límite para el tamaño del bloque.
• Conmutación de paquetes: Aquí los datos se dividen en fragmentos llamados paquetes que pueden viajar por múltiples rutas entre distintas computadoras. Como los paquetes pueden viajar en ambas direcciones, requieren una dirección destino. La conmutación de paquetes no reserva ancho de banda y lo adquiere conforme lo necesita, por lo que es muy útil en el manejo del tráfico interactivo. Aquí los paquetes son guardados en la memoria de las centrales de conmutación.
• Conmutación híbrida: Son las variantes que pueden existir en la conmutación de circuitos y paquetes que tratan de aprovechar las ventajas de cada una, como la conmutación de circuitos por conexión rápida y la conmutación por división de tiempo.

2.4 Tipos de redes

Las redes de cómputo, que integran diversos componentes que operan entre sí para compartir recursos, pueden clasificarse de diferentes maneras [Black, 1993]. Los tipos de redes de cómputo más conocidos son:

a) Por su servicio:
• Redes públicas, redes de acceso público, por ejemplo, la red telefónica pública comercial.
• Redes privadas, utilizadas dentro del ámbito de una empresa para su uso exclusivo.

b) Por su funcionamiento:

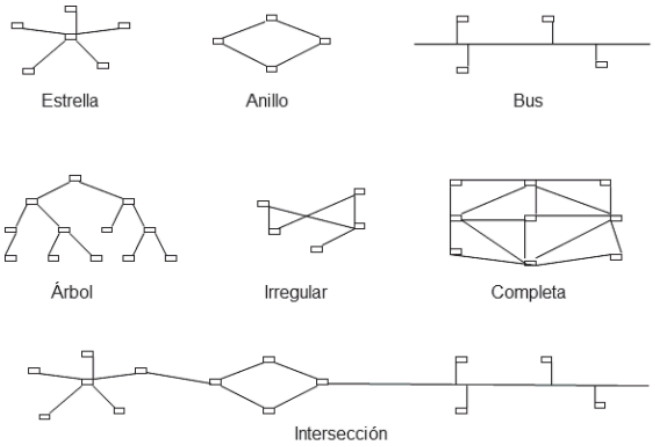
• Redes de conmutación, se usan en los métodos de conmutación para ofrecer la comunicación entre dispositivos (teléfonos o computadoras personales).
• Redes de difusión (Broadcast), redes en que la comunicación se realiza por difusión desde un dispositivo a más de un dispositivo (señales de satélite, radio y TV).

c) Por su extensión:
• Redes de área local (LAN), redes con múltiples usuarios conectados en una parte geográfica pequeña, por lo general no mayor a 1 km; ofrecen canales de comunicación de alta velocidad y por lo general son redes privadas y relativamente libres de error.
• Redes de área metropolitana (MAN), redes con múltiples usuarios conectados en una parte geográfica que va de 1 a 10 km, aproximadamente.
• Redes de área amplia (WAN), redes con múltiples usuarios conectados en una amplia región geográfica, como entre ciudades, países o continentes, por lo general la comunicación abarca líneas telefónicas, satélites y redes de cómputo, sus canales son relativamente de baja capacidad y de mayor margen de error.

2.5 Topología de redes

La topología de redes se refiere al arreglo geométrico que tendrán las conexiones entre las computadoras, además, son una forma para clasificar las redes. Las topologías más usuales que se muestran en la figura 2.2 son:

• Topología estrella: Todas las computadoras se conectan a una computadora central. Esta topología no permite la comunicación directa entre dos computadoras que no sean la central.
• Topología en anillo: La red forma un anillo continuo en el cual puede viajar la información.
• Topología en bus: Emplea un solo medio llamado bus, al cual todas las computadoras se conectan.
• Topología en árbol: Existe una computadora principal que sirve como raíz y única salida externa.
• Topología irregular: No se respeta un modelo de conexión.
• Topología de intersección: En esta topología existe la conexión de dos o más tipos de topologías.
• Topología completa: Se dan todos los tipos de topologías.
Figura 2.2. Topologías más usadas para las redes de computadoras

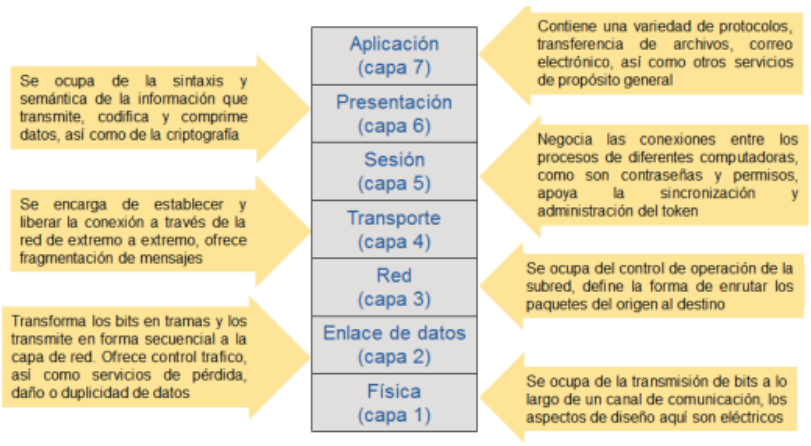


2.6 Modelo OSI

El advenimiento de los sistemas de cómputo propició un campo ideal para que los sistemas fueran abiertos, es decir que diferentes plataformas de diferentes fabricantes se pudieran comunicar. Sin embargo, antes de la década de 1970 esto no era posible porque los sistemas eran cerrados y los fabricantes seguían sus propios modelos de comunicación. Para afrontar este problema, entre 1977 y 1984 los especialistas del ISO (International Standard Organization) crearon el OSI (Open System Interconnection), un modelo de referencia para la interconexión de sistemas abiertos. El modelo ISO/OSI utiliza siete capas para organizar una red en módulos funcionales bien definidos (ver figura 2.3), con la cual los diseñadores puedan construir redes reales, sin embargo, al ser este solo un modelo y no una norma, el diseñador puede modificar el número, nombre y función de la red. Los principios aplicados para el establecimiento de siete capas fueron [Tanenbaum, 1997]:

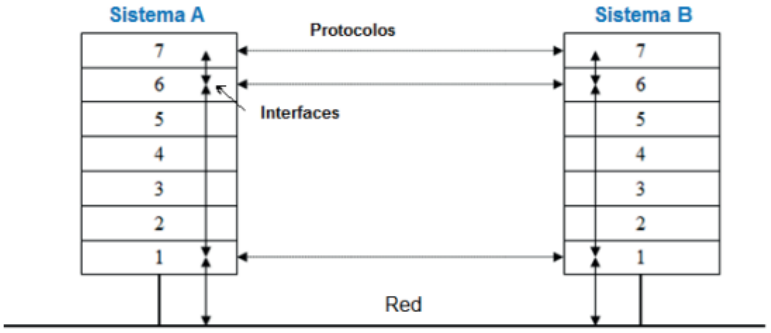
1. Una capa se creará en situaciones en las que se necesita un nivel diferente de abstracción.
2. Cada capa deberá efectuar una función bien definida.
3. La función que realizará cada capa deberá seleccionarse con la intención de definir protocolos normalizados internacionalmente.
4. Los límites de las capas deberán seleccionarse tomando en cuenta la minimización del flujo de información a través de las interfaces.
5. El número de capas deberá de ser lo suficientemente grande para que funciones diferentes no tengan que ponerse juntas en la misma capa y, por otra parte, también deberá de ser lo suficientemente pequeño para que su arquitectura no sea difícil de manejar.

Figura 2.3. Las capas del modelo OSI y su aplicación



Una vez establecido el modelo OSI, las computadoras se pueden comunicar como se ilustra en la figura 2.4. La comunicación se realiza de una capa de un nivel a otra capa del mismo nivel por medio de protocolos (líneas punteadas). Se usan interfaces para transmitir los datos de una capa a otra capa inferior dentro de una misma computadora; así, hasta llegar a la capa física y pasar a la red subiendo a la capa física de la otra computadora.

Figura 2.4. Comunicación entre dos computadoras usando el modelo OSI



2.7 Norma IEEE-802

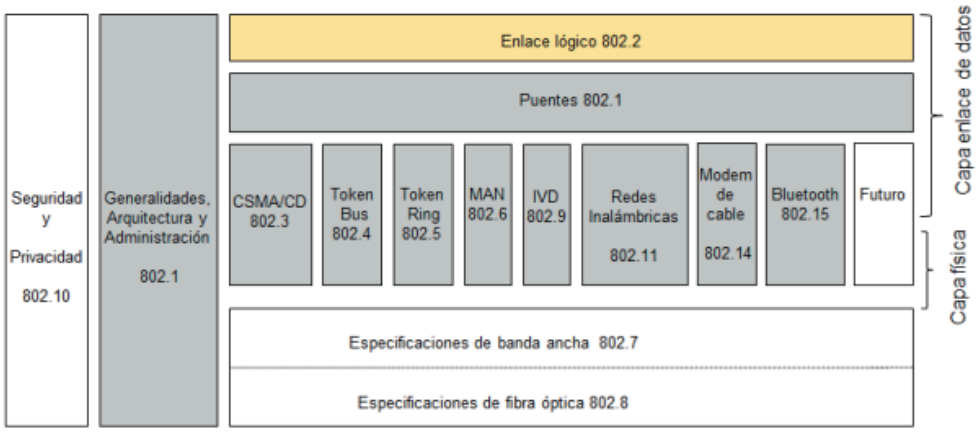
El estándar IEEE-802 del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) específica las características que deben de cumplir las redes LAN y define las especificaciones para cada uno de estos, como se indica en la tabla 2.2 (sólo se muestran los estándares iniciales).

Tabla 2.2. Componentes del estándar IEEE-802 para LAN [Black, 1993]
Nombre del estándar Descripción
802.1 Da una introducción al conjunto de normas y define las primitivas de interfaz
802.2 Describe la parte superior de la capa de enlace, la LLC
802.3 Especificaciones para la tecnología de acceso CSMA/CD (para Ethernet)
802.4 Especificaciones para la tecnología de testigo en bus
802.5 Especificaciones para la tecnología de testigo en anillo
802.6 Especificaciones para redes de área metropolitana

802.7 Especificaciones para redes de banda ancha
802.8 Especificaciones para redes de fibra óptica
802.9 Especificaciones para redes que integran datos y voz
802.10 Especificaciones para seguridad en redes
802.11 Especificaciones para redes inalámbricas

La arquitectura del estándar 802 se muestra en la figura 2.5.

Figura 2.5. El estándar IEEE 802 [adaptado de Black, 1993]



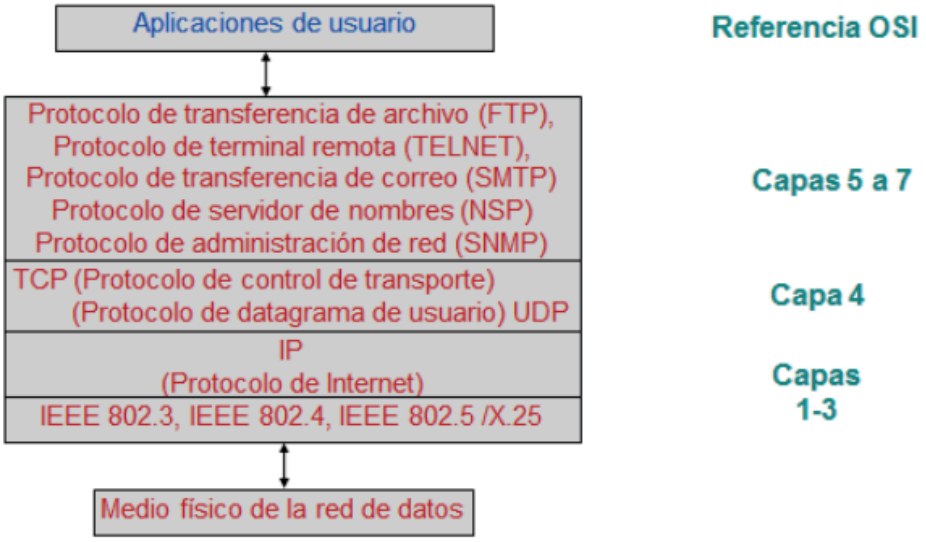
CSMA/CD: Acceso múltiple por detección de portadora con detección de colisión
MAN: Red de área metropolitana
IVD: Integración de voz y datos

2.8 Protocolos y paquetes

2.8.1 Protocolos

Los protocolos son un conjunto de reglas que gobiernan la interacción de procesos concurrentes en sistemas distribuidos, estos son utilizados en un gran número de campos como sistemas operativos, redes de computadoras o comunicación de datos.
Uno de los conjuntos de protocolos más usados en Internet y que usamos en los ejemplos de sockets es el TCP/IP (Transmission Control Protocol / Internet Protocol). Este conjunto de protocolos tiene menos capas que el OSI, lo que incrementa su eficiencia. TCP/IP es un protocolo confiable, ya que los paquetes son recibidos en el orden en que son enviados. Muchos
sitios también usan el protocolo UDP/IP (User Datagrama Protocol/Internet Protocol). El protocolo UDP/IP es un protocolo no confiable, ya que no garantiza la entrega. La figura 2.6 ilustra las capas del protocolo TCP/IP y UDP/IP comparado con el modelo OSI.

Figura 2.6. Arquitectura de los protocolos TCP/IP y UDP/IP

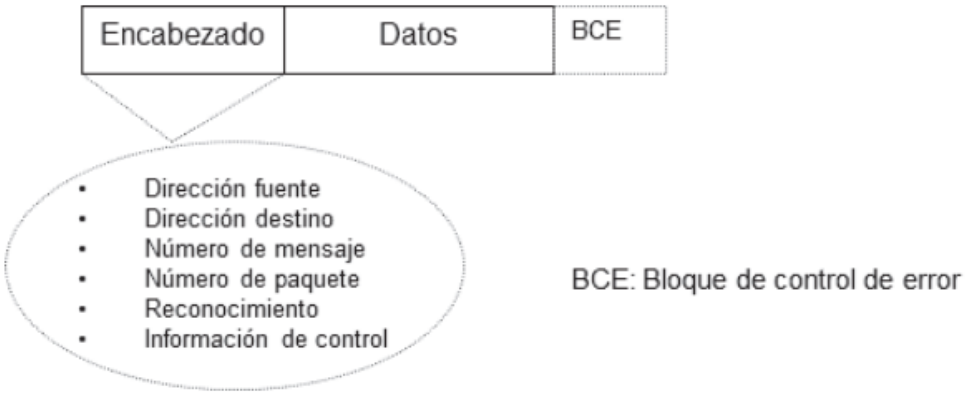


2.8.2 Paquetes

Un paquete es la forma usada para enviar información en un ambiente distribuido o de red. Cada mensaje es dividido y colocado en paquetes. Un paquete contiene toda la información necesaria para construir el mensaje original. Es decir que los paquetes pueden llegar en desorden, pero el nodo destino deberá de ser capaz de poner los paquetes en una forma ordena.
El segmento de datos del paquete contiene (ver figura 2.7):

• Los encabezados de cada capa del protocolo a partir de los datos de la capa de enlace.
• Los datos de la aplicación actual.

Figura 2.7. Formato típico de un paquete



Cuando los paquetes llegan a la capa de enlace de datos (capa 2 del modelo OSI), son entramados en un marco (trama) donde se adiciona un bloque de control de error. Aunque a veces se usan los términos de paquetes y marcos como equivalentes, esto no es correcto, ya que los términos se refieren a unidades en diferentes capas del modelo OSI (la de

transporte y la de enlace de datos, respectivamente). Los estándares IEEE 802.3, IEEE-802.4 e IEEE 802.5, tienen su propio formato de marco para sus paquetes.

2.8.3 Redes de banda ancha

Muchas de las tecnologías emergentes se designan como redes multimedia de banda ancha. El propósito de una red de multimedia de banda ancha es proporcionar un servicio de transporte para cualquier tipo de aplicación. Una red de banda ancha apoya el servicio telefónico, de video, de datos, compras a distancia, aplicaciones CAD/CAM, etc. De acuerdo con Black [1999], el término banda ancha ha sido usado desde hace varios años y se han propuesto algunas definiciones para la tecnología de banda ancha, como las siguientes:
• “Cualquier sistema de alta capacidad que ofrece un sistema de transporte de multimedia.
• Una red que utiliza tecnología de alta frecuencia como mecanismo de transporte para el tráfico de usuarios.
• Cualquier red que opere por encima del intervalo de frecuencia de la voz (0-4 kHz).
• Cualquier red que opere por arriba de la tasa primaria de ISDN (1.544 Mbit/s en Norteamérica y 2.048 Mbit/s en Europa)”.

Dentro de las redes de banda ancha, la tecnología de red más popular es la tecnología ATM (Asynchronous Transfer Mode). La tecnología ATM está basada en los esfuerzos del grupo de estudio XVIII del ITU-T, al desarrollar el BISDN para la transferencia a alta velocidad de voz, video y datos a través de redes públicas.

El propósito de ATM es proporcionar una red con multiplexión y conmutación, alta velocidad y bajo retardo para apoyar cualquier tipo de tráfico de usuario, como aplicaciones de voz, datos o video [Ford, Lew, Spanier & Stevenson, 1998]. ATM segmenta y multiplexa el tráfico de usuario en unidades pequeñas de longitud fija llamadas celdas. La celda tiene 53 bytes, de los cuales 5 están reservados para el encabezado de la celda, los 48 bytes restantes se ocupan para datos. Cada celda se identifica con identificadores de circuitos virtuales contenidos en el encabezado. Una red ATM utiliza esos identificadores para encaminar el tráfico a través de computadoras de alta velocidad desde el equipo de las instalaciones del cliente (CPE) transmisor hasta el CPE receptor. ATM ofrece operaciones de detección de error limitada. Con la excepción del tráfico de señalización, ATM no ofrece servicios de retransmisión y son pocas las operaciones que se realizan con el pequeño encabezado. La intención de tener celdas con pocos servicios prestados es implementar una red que sea lo bastante rápida como para apoyar tasas de transferencia.

2.8.4 Redes inalámbricas

Actualmente la movilidad y el cómputo móvil emergen con gran fuerza. Millones de personas intercambian información cada día usando receptores de mensajes, tabletas electrónicas, teléfonos móviles y otros productos de comunicación inalámbrica. Con el auge de la telefonía inalámbrica y los servicios de mensajería, es una tendencia el que la comunicación inalámbrica se aplique al campo de la computación personal y de los

negocios. La gente desearía tener facilidades de acceso y compartir información en una escala global y cercana en cualquier sitio en que se encuentre. El propósito de un sistema de comunicaciones móviles se puede inferir del nombre de la tecnología, que es la de prestar servicios de telecomunicaciones entre estaciones móviles y estaciones terrenas fijas, o entre dos estaciones móviles [Black, 1999]. Se pueden distinguir dos formas de comunicación móviles: celular e inalámbrica.

Ejercicios

1. ¿Cuál es la diferencia entre una red LAN y MAN?
    - Diferencia entre LAN y MAN: Una LAN (Red de Área Local) conecta dispositivos en un área geográfica pequeña, como una oficina o un edificio, mientras que una MAN (Red de Área Metropolitana) cubre un área más grande, generalmente una ciudad o un campus universitario.
2. ¿Cuál es la función del protocolo IEEE 802.11?
    - Función del protocolo IEEE 802.11: El protocolo IEEE 802.11 se utiliza para redes inalámbricas, proporcionando estándares para la comunicación de dispositivos Wi-Fi.
3. ¿Qué ventajas tiene usar celdas en comparación con usar paquetes en la comunicación de datos?
    - Ventajas de usar celdas en comparación con paquetes: Las celdas ofrecen un enfoque más predecible para la transmisión de datos, ya que tienen un tamaño fijo y un tiempo de transmisión constante, lo que simplifica la gestión del tráfico y reduce la complejidad de los dispositivos de red.
4. ¿En qué consiste el sistema GSM y cuáles son sus principales componentes?
    - Sistema GSM y sus principales componentes: GSM (Sistema Global para Comunicaciones Móviles) es un estándar para la telefonía móvil digital. Sus principales componentes son estaciones base, conmutadores de red, estaciones móviles y la red de operador.
5. ¿Cuál es la similitud entre una topología en árbol y una de estrella?
    - Similitud entre topología en árbol y topología en estrella: Ambas topologías tienen un nodo central al que se conectan otros nodos, permitiendo una comunicación centralizada y organizada.
6. Investiga las características de los medios físicos de comunicación para redes de cómputo.
    - Características de los medios físicos de comunicación para redes de cómputo: Los medios físicos incluyen cables de cobre, fibra óptica y transmisión inalámbrica. Cada uno tiene diferentes velocidades, distancias máximas de transmisión y susceptibilidad a interferencias.
7. ¿Qué esquemas de comunicación se utilizan para las redes satelitales?
    - Esquemas de comunicación para redes satelitales: Los esquemas incluyen transmisión unidireccional, bidireccional y multidifusión, adaptados para comunicaciones de larga distancia y cobertura global.
8. En el modelo OSI, ¿cuál es la diferencia entre un protocolo y una interfaz?

- Diferencia entre protocolo e interfaz en el modelo OSI: Un protocolo define las reglas para la comunicación entre capas, mientras que una interfaz proporciona los puntos de acceso entre capas adyacentes.

9. ¿Qué tipos de redes existen en la Internet?
    - Tipos de redes en Internet: Incluyen redes LAN, WAN, MAN, redes inalámbricas, redes de área personal (PAN), entre otras.
10. ¿Cuál es la principal desventaja de una topología en anillo?
    - Principal desventaja de una topología en anillo: Si un nodo falla, puede afectar la comunicación en toda la red.
11. ¿Qué beneficios aporta usar una topología de árbol en una red de difusión de contenidos?
    - Beneficios de usar una topología de árbol en una red de difusión de contenidos: Permite una distribución eficiente de la información a múltiples nodos, evitando congestiones en la red.
12. ¿Qué beneficios aporta usar una topología completa o de malla en una red de datos?
    - Beneficios de usar una topología completa o de malla en una red de datos: Proporciona redundancia y rutas alternativas, lo que mejora la fiabilidad y la tolerancia a fallos de la red.
13. Considera que dos computadoras transmiten paquetes de 1,500 bytes por un canal compartido que opera a 128,000 bits por segundo. Si las computadoras tardan 90 microsegundos entre la terminación de transmisión de una computadora y el inicio de otra, ¿qué tiempo se requiere para que una computadora envíe un archivo de 5 MB a la otra computadora?
    - Por lo tanto, se requerirían aproximadamente 312.97 segundos para que una computadora envíe un archivo de 5 MB a la otra computadora.

Capítulo 3. Modelos de arquitecturas

Objetivo: Que el alumno comprenda los diferentes paradigmas de cómputo, desde el cliente-servidor al grid, y cómo se integran en los modelos arquitectónicos de los sistemas distribuidos.

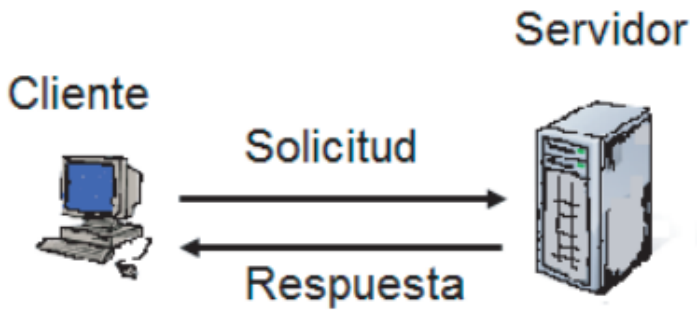
3.1 Introducción

La arquitectura de un sistema es su estructura en términos de los componentes especificados por separado y sus interrelaciones. El objetivo de una arquitectura general es asegurar que la estructura reunirá presentes y probables futuras demandas sobre el mismo. Las principales preocupaciones son que el sistema sea fiable, manejable, adaptable y rentable [Coulouris et al., 2012]. La arquitectura de un sistema distribuido guarda algunos aspectos similares con el diseño arquitectónico de un edificio, los cuales determinan no solo su apariencia, sino también su estructura general y el estilo arquitectónico (gótico, neoclásico y moderno) y proporciona un marco coherente de referencia para el diseño. Todos los tipos de sistemas distribuidos tienen características básicas comunes. Un modelo de arquitectura es una descripción abstracta simplificada pero consistente de cada aspecto relevante del diseño de un sistema distribuido. Este capítulo describe los principales modelos arquitectónicos de los sistemas distribuidos, particularmente en paradigmas cliente-servidor, peer-to-peer, grid, proxy, cluster y las principales diferencias entre estos.

3.2 Modelo cliente - servidor

El modelo cliente-servidor es la arquitectura más citada cuando se discuten los sistemas distribuidos. Es el modelo más importante y sigue siendo el más ampliamente utilizado. La figura 3.1 ilustra la estructura simple de esta arquitectura, en la cual los procesos toman el rol de ser clientes o servidores. En particular, los procesos de cliente interactúan con los procesos de servidor individuales en equipos anfitriones (host) potencialmente separados, con el fin de acceder a los recursos compartidos que administran.

Figura 3.1. Ejemplo de una estructura simple cliente-servidor



El modelo cliente-servidor puede tomar diferentes configuraciones. Por ejemplo, puede existir más de un cliente conectado a un servidor, como se indica en la figura 3.2. También se puede tener un grupo de servidores interconectados dedicados a dar servicio a un grupo de clientes. Este escenario se indica en la figura 3.3.

Figura 3.2. Ejemplo de estructura cliente-servidor para dos clientes

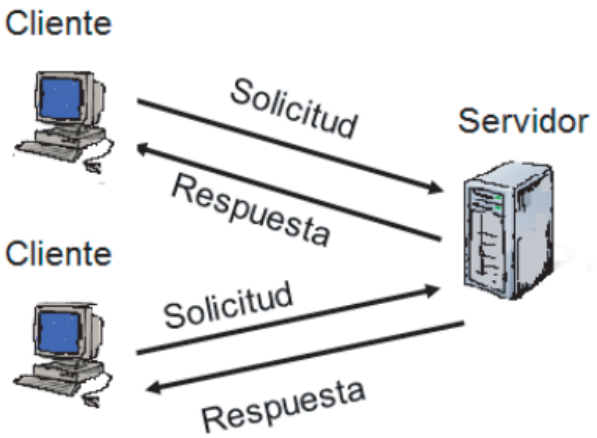
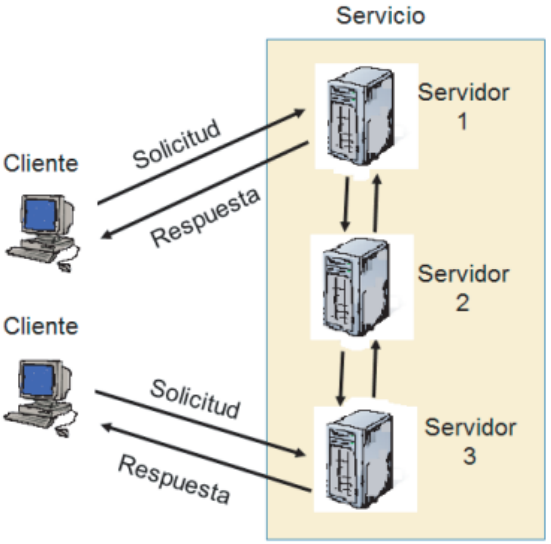


Figura 3.3. Grupo de servidores interconectados basado en el modelo cliente-servidor



3.3 Proxy

Es un servidor que se emplea como intermediario entre las peticiones de recursos que realiza un cliente a otro servidor. Por ejemplo, si una computadora A solicita un recurso a una computadora C, lo hará mediante una petición a la computadora B que, a su vez, trasladará la petición a la computadora C. De esta manera, la computadora C no sabrá que

la petición procedió originalmente de la computadora A. Esta situación estratégica de punto intermedio suele ser aprovechada para soportar una serie de funcionalidades, como:
• Proporcionar caché.
• Control de acceso.
• Registro del tráfico.
• Prohibir cierto tipo de tráfico.
• Mejorar el rendimiento.
• Mantener el anonimato.

El proxy más conocido es el servidor proxy web, su función principal es interceptar la navegación de los clientes por páginas web por motivos de seguridad, rendimiento, anonimato, entre otros. Las figuras 3.4 y 3.5 muestran dos ejemplos del uso de proxy.

Figura 3.4. Arreglo de proxy cliente y proxy servidor para acceder al servidor desde dos clientes

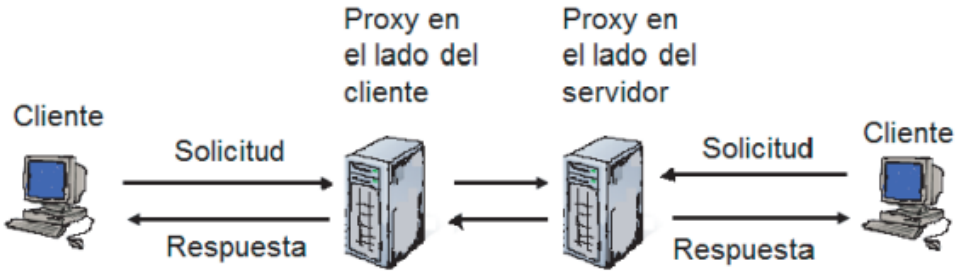
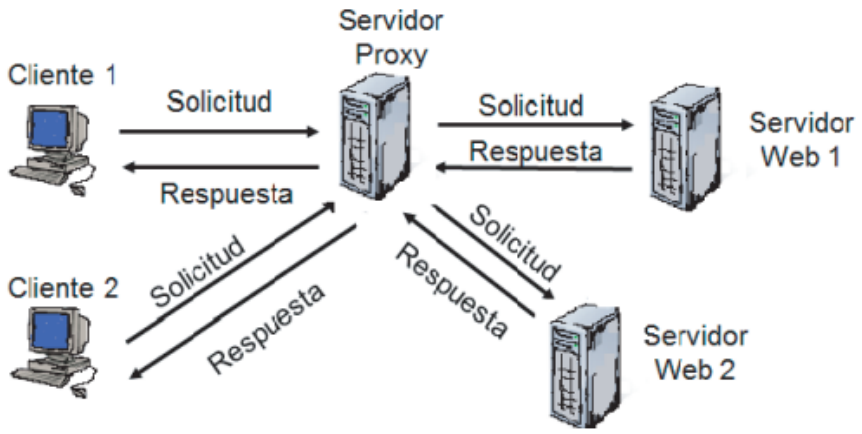


Figura 3.5. Acceso a servidores web vía un proxy



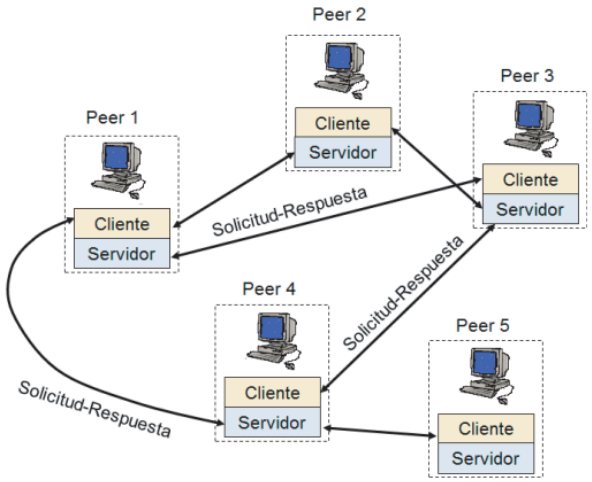
3.4 Peer-to-Peer

El paradigma peer-to-peer (P2P) ha sido un tema muy atractivo para muchos investigadores de diferentes áreas, tales como redes, sistemas distribuidos, teoría de la complejidad, bases de datos y otros. En el modelo cliente-servidor tradicional, dos tipos de nodos son empleados: clientes y servidores. En este contexto, los clientes solo solicitan servicios y el

servidor solo proporciona a los clientes el servicio apropiado. Un servidor puede aceptar varias solicitudes, procesarlas y devolver los contenidos solicitados a los clientes. En la Internet actual, los clientes incluyen navegadores web, clientes de chat en línea y clientes de correo electrónico, mientras que los servidores normalmente son servidores web, servidores FTP y servidores de correo. En contraste, en los sistemas P2P no se requiere una infraestructura dedicada. Los servidores dedicados y clientes no existen, ya que cada peer puede tomar el papel tanto de servidor como de cliente al mismo tiempo. Una ventaja importante de los sistemas peer-to-peer es que todos los recursos disponibles son proporcionados por los peers. Durante la distribución de un contenido, los peers aportan sus recursos para transmitir el contenido a los demás peers. Por lo tanto, cuando un nuevo peer se agrega al sistema al sistema P2P, la demanda se incrementa pero la capacidad general del sistema también. Esto no es posible en un modelo cliente-servidor con un número fijo de servidores.
El paradigma P2P se muestra en la figura 3.6, donde se puede ver que en un peer están ejecutándose al mismo tiempo tanto un proceso servidor como uno cliente, también ambos ofrecen y solicitan respectivamente servicios a otros procesos similares en otros peers.

Beneficios de un sistema peer-to-peer:
• Nodos comparten recursos.
• Se pueden desplegar algoritmos distribuidos.
• Escalamiento más fácil del sistema.
• Ahorro de costos.
• Flexibilidad.
• Ningún punto único de falla.
• Mayor robustez del sistema.

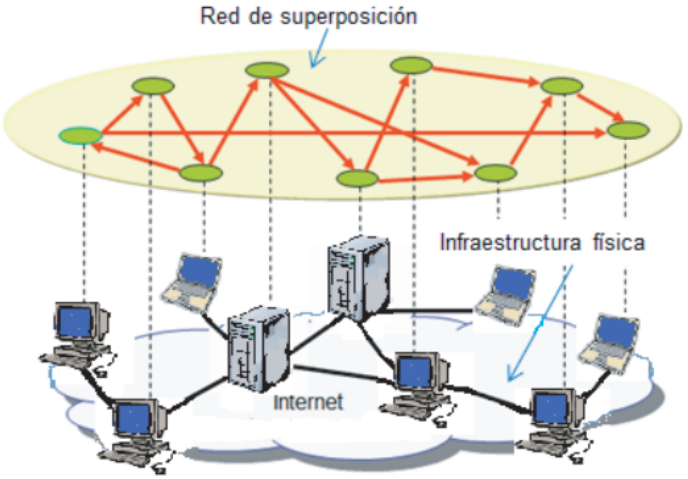
Figura 3.6. Paradigma peer-to-peer



Una infraestructura de comunicación P2P está formada por un grupo de nodos ubicados en una red física. Estos nodos construyen una abstracción de red en la parte superior de la red física conocida como red superpuesta, que es independiente de la red física subyacente. La figura 3.7 muestra este escenario. La red superpuesta se establece para cada sistema P2P a través de conexiones TCP o HTTP. Debido a la capa de abstracción de la pila de

protocolo TCP, las conexiones físicas no son reflejadas por la red de superposición. La red superpuesta construye túneles lógicas entre pares de nodos [Ripeanu, Foster, Iamnitchi & Rogers, 2007], con el fin de implementar su propio mecanismo de enrutamiento para transportar sus mensajes.

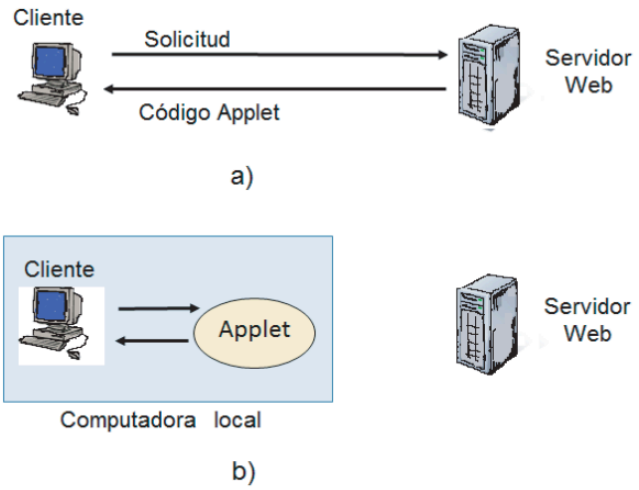
Figura 3.7. Ejemplo de una red superpuesta peer-to-peer [López-Fuentes, 2009]



3.5 Applets

Un applet es un código que se ejecuta en el contexto de otro programa, por ejemplo, en un navegador web. El código se descarga en el navegador y se ejecuta allí, como se muestra en la figura 3.8.

Figura 3.8. a) A solicitud del cliente el servidor web, responde con el código del applet. b) El cliente interactúa con el applet (adaptado de [Coulouris et al., 2012])



Un applet normalmente lleva a cabo una función muy específica, que carece de uso independiente, y son ampliamente utilizados en aplicaciones de telefonía móvil. Un applet puede dar una buena respuesta interactiva, ya que no sufre de los retrasos o variabilidad de ancho de banda asociado con la comunicación de la red. Sin embargo, un applet típicamente carece de sesión y tiene privilegios restringidos de seguridad. A menudo, un applet consiste en un código poco confiable, por eso se les impide tener acceso al sistema de archivos local. Los applet que se cargan a través de la red con frecuencia son considerados como códigos de poca confianza [Flanagan, 1998], a excepción de que lleven la firma digital de una entidad especificada como confiable. Ejemplos de los applets más comunes son:
• Java applets.
• Animaciones Flash.
• Windows media player.
• Modelos 3D.

3.6 Clúster

En informática, el término clúster (“grupo” o “racimo”) hace referencia a conjuntos o conglomerados de computadoras construidos mediante el uso de hardware común y que se comportan como si fueran una única computadora. Un ejemplo de clúster se muestra en la figura 3.9. El uso de los clústeres varía desde las aplicaciones de supercómputo, servidores web y comercio electrónico hasta el software de misiones críticas y bases de datos de alto rendimiento. El cómputo con clústeres es el resultado de la convergencia de varias tendencias tecnológicas actuales, entre las que se pueden destacar:
• Microprocesadores de alto rendimiento.
• Redes de alta velocidad.
• Software para cómputo distribuido de alto rendimiento.
• Crecientes necesidades de potencia computacional.

Los servicios esperados de un clúster principalmente son:
• Alto rendimiento.
• Alta disponibilidad.
• Escalabilidad.
• Balanceo de carga.

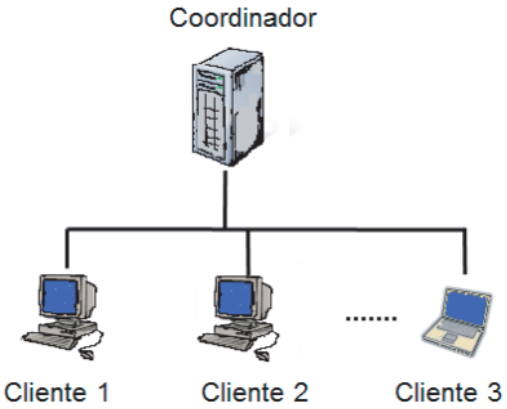
Típicamente respecto a la rapidez y disponibilidad, se espera que un clúster sea más económico que el uso de computadoras individuales.
Un clúster puede ser:

• Homogéneo.
• Semihomogéneo.
• Heterogéneo.

Un clúster es homogéneo cuando todas las computadoras tienen la misma configuración en hardware y sistema operativo. Es semihomogéneo cuando las computadoras tienen diferente rendimiento pero guardan una similitud con respecto a su arquitectura y sistema

operativo. Finalmente, un clúster es heterogéneo cuando las computadoras tienen diferente hardware y sistema operativo.

Figura 3.9. Ejemplo de clúster



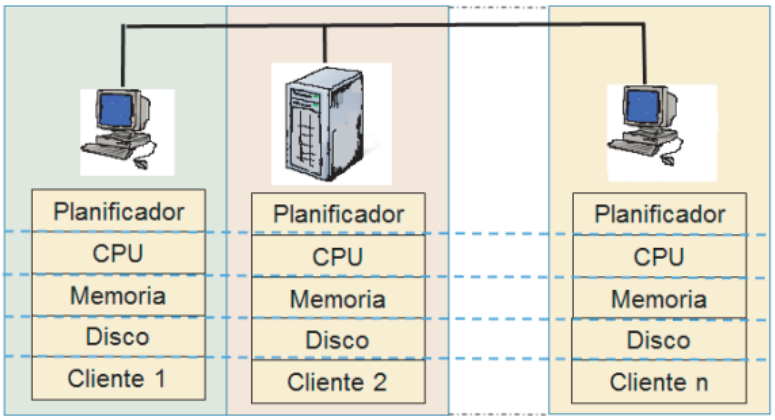
3.7 Grid

El cómputo grid es un paradigma del cómputo distribuido, frecuentemente usado para indicar una infraestructura de gestión de recursos distribuidos que se centra en el acceso coordinado a los recursos informáticos remotos [Foster & Kesselman, 1999]. Estos recursos de cómputo son colectados desde múltiples localizaciones para alcanzar una meta común. A diferencia del cómputo de cluster (en grupo o racimo), el cómputo grid tiende a ser más heterogéneo y disperso geográficamente. Generalmente las grids son usadas para una variedad de propósitos pero puede haber grids especializadas para fines específicos. Los recursos que son integrados por una infraestructura grid son típicamente plataformas de cómputo dedicadas a supercomputadoras de alta gama o clústers de propósito general. Algunos autores [Foster & Kesselman, 2013] ubican como antecedente del cómputo grid al sistema de tiempo compartido Multics. Sin embargo, este concepto ha sufrido múltiples transformaciones a lo largo de los años y diversas definiciones sobre el cómputo grid pueden ser encontradas en la literatura. Jacob, Brown, Fukui y Trivedi [2005] definen la computación grid como cualquiera de una variedad de niveles de virtualización a lo largo de un continuo, donde a lo largo de ese continuo se podría decir que una solución particular es una implementación de cómputo grid frente a una relativamente simple aplicación usando recursos virtuales, pero incluso en los niveles más simples de virtualización, siempre se requieren habilitar tecnologías de redes.

Beneficios del cómputo grid [Jacob et al., 2005]:
• Explotación de recursos infrautilizados.
• Capacidad de CPU paralelos.
• Recursos virtuales y organizaciones virtuales para la colaboración.
• Acceso a recursos adicionales.
• Balanceo de recursos.
• Fiabilidad.
• Mejor gestión de infraestructuras de TI más grandes y distribuidos.

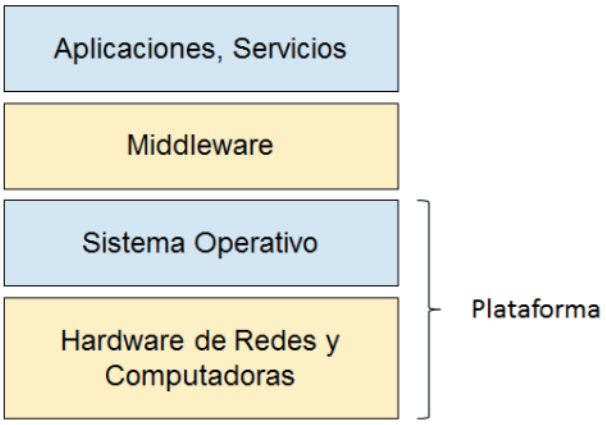
La conceptualización del cómputo grid se muestra en la figura 3.10.

Figura 3.10. Ejemplo de cómputo grid



3.8 Arquitectura de capas

Una arquitectura de capa resulta familiar en los sistemas distribuidos y está relacionado con la abstracción. Con este enfoque, un sistema complejo puede ser dividido en cierto número de capas, donde las capas superiores hacen uso de los servicios ofrecidos por las capas inferiores. De esta manera, una determinada capa ofrece una abstracción de software, sin que las capas superiores o inferiores a esta deban de estar al tanto de los detalles de implementación. En el caso de los sistemas distribuidos, los servicios se organizan de una manera vertical como capas de servicio. Un servicio distribuido puede ser proporcionado por uno o más procesos del servidor, que interactúan entre sí y con los procesos de cliente para mantener una visión de todo el sistema, coherente de los recursos del servicio. Por ejemplo, el ajuste de hora entre clientes puede realizarse por un servicio de hora de red a través de Internet, usando el protocolo de Tiempo de Red (NTP). Es útil organizar este tipo de servicio en capas debido a la complejidad de los sistemas distribuidos. Una visión común de una arquitectura de capa se muestra en la figura 3.11.
Figura 3.11. Capas de servicio en un sistema distribuido [Coulouris et al., 2012]



Como se indica en la figura 3.11, un sistema distribuido está constituido principalmente por los siguientes estratos:
• Plataforma.
• Middleware.
• Aplicaciones y servicios.

La plataforma para sistemas y aplicaciones distribuidas se compone de las capas de hardware y software de nivel más bajo. Estas capas de bajo nivel proporcionan servicios a las capas superiores, las cuales son implementadas de manera independiente en cada equipo, conduciendo a la interfaz de programación del sistema hasta un nivel que facilita la comunicación y la coordinación entre los procesos. Ejemplos de plataformas son:

• Intel x86/Windows.
• Intel x86/Mac OS X.
• Intel x86/Linux.
• Intel x86/Solaris.

Middleware es un software que tiene como función principal enmascarar la heterogeneidad del sistema distribuido para proporcionar un modelo de programación conveniente a los programadores de aplicaciones. Ejemplos de middleware son:

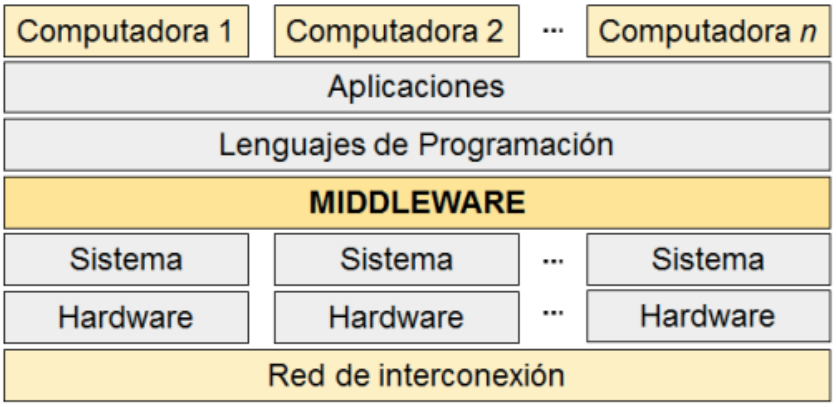
• CORBA (Common Object Request Broker).
• Java RMI (Java Remote Method Invocation).
Finalmente, la capa de aplicaciones y servicios son las prestaciones que ofrece el sistema distribuido a los usuarios. Se entiende como las aplicaciones distribuidas.

3.9 Middleware

En la primera generación de los sistemas distribuidos todos los servicios proporcionados por los servidores debían de programarse a la medida. Así, servicios de acceso a bases de datos, de impresión y transferencias de archivos tenían que ser desarrollados por las propias aplicaciones. Queda en evidencia la necesidad de crear servicios de uso más común por las aplicaciones, de tal manera que pueda incluirse en todas las aplicaciones como software prefabricado. En este escenario surge la idea del middleware, representado por estándares tales como ODBC, OLE, DCOM y CORBA.
Middleware es un conjunto de servicios que permite distribuir datos y procesos a través de un sistema multitarea, una red local, una red remota o Internet [Martínez Gomaríz, 2014].
Los servicios del Middleware pueden ser clasificados en dos grandes grupos:
• Servicios de desarrollo.
• Servicios de administración.
El objetivo principal del middleware es conseguir la transparencia en los sistemas distribuidos, por medio de:
• Ofrecer la capacidad, así como solicitar y recibir de manera transparente al sistema.
• Liberar a los diseñadores y administradores del sistema de problemas derivados de la complejidad del sistema operativo.

En la práctica, middleware es representado por procesos u objetos en un conjunto de equipos que interactúan entre sí para implementar la comunicación y el intercambio de recursos de soporte para las aplicaciones distribuidas [Coulouris et al.,2012]. El middleware está relacionado con el suministro de materiales de construcción útiles para la construcción de componentes de software que pueden trabajar con otros en un sistema distribuido. Las abstracciones del middleware apoyan a diversas actividades de comunicación, como la invocación de método remoto, la comunicación entre un grupo de procesos, notificación de eventos, el particionamiento, la colocación y recuperación de objetos de datos compartidos entre los equipos cooperantes, la replicación de objetos de datos compartidos y la transmisión de datos multimedia en tiempo real. Un escenario del middleware es mostrado en la figura 3.12.

Figura 3.12. Escenario del middleware en un sistema distribuido



A pesar de que el middleware tiene como objetivo suministrar transparencia de distribución, en general las soluciones específicas deben de ser adaptables a los requerimientos de las aplicaciones. Tanenbaum y Van Steen [2008] consideran que una solución para este problema es desarrollar diversas versiones de un sistema middleware, donde cada versión sea confeccionada para una clase específica de aplicaciones.

3.10 CORBA

El paradigma orientado a objetos juega un importante rol en el desarrollo de software y cuenta con gran popularidad desde su introducción. La orientación a objetos se comenzó a utilizar para el desarrollo de sistemas distribuidos en la década de 1980. El Grupo de Gestión de Objetos (OMG-Object Management Group) se creó en 1989 como una asociación de las empresas líderes de la tecnología software para definir especificaciones que puedan ser implementadas por ellas y que faciliten la interoperabilidad de sus productos. Los middlewares basados en componentes se han convertido en una evolución natural de los objetos distribuidos, debido a que apoyan la gestión de las dependencias entre componentes, ocultando los detalles de bajo nivel asociados con el middleware, gestión de las complejidades de establecer aplicaciones distribuidas con propiedades no funcionales apropiadas –como la seguridad– y el apoyo apropiado de estrategias de

implementación. Los middlewares basados en objetos distribuidos están diseñados para proporcionar un modelo de programación basado en principios orientados a objetos y, por tanto, para llevar los beneficios del enfoque a objetos para la programación distribuida. Los principales ejemplos de middleware basados en objetos distribuidos incluyen Java RMI y CORBA.
CORBA (Common Object Request Broker Architecture) es una herramienta middleware que facilita el desarrollo de aplicaciones distribuidas en entornos heterogéneos tanto en hardware como en software [Coulouris et al., 2012], ejemplos de estos son:
• Distintos sistemas operativos (Unix, Windows, MacOS, OS/2).
• Distintos protocolos de comunicación (TCP/IP, IPX).
• Distintos lenguajes de programación (Java, C, C++).
• Distinto hardware.

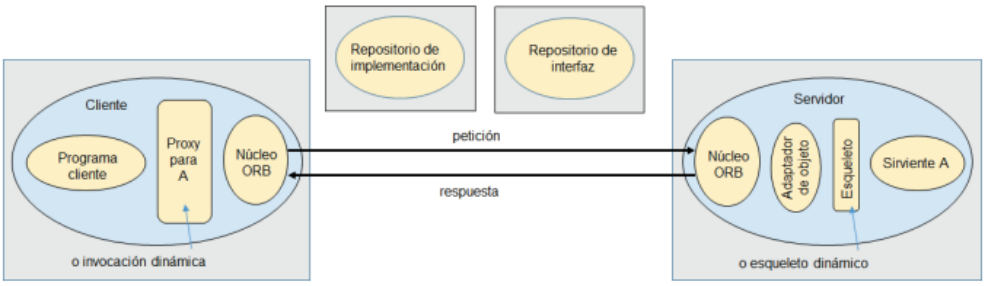
El objetivo principal de CORBA es especificar un middleware para construir aplicaciones del tipo cliente-servidor multi-nivel, distribuidas y centralizadas, y que sean flexibles y extensibles. Para alcanzar estos objetivos, CORBA utiliza diferentes medios, como los que a continuación se describen.

Del paradigma orientado a objeto explota los conceptos de encapsulación, herencia y polimorfismo. En la comunicación a través de invocar métodos remotos es más fácil de programar que los sockets, RPC, etc. Usa el concepto de esqueleto como el encargado de la comunicación con el cliente. Define la separación interfaz-implementación a través de CORBA IDL (Interface Definition Language). Para la referencia a objeto, usa el identificador único de un objeto. CORBA es implementado encima del protocolo TCP/IP y usa modos de comunicación asíncrona y síncrona. Con el uso de envolturas (wrappers), CORBA permite integrar los sistemas heredados, y normalmente usa solo una instancia de cada clase. CORBA también hace uso de una capa de software conocida como ORB (Object Request Broker), que permite a los objetos realizar llamadas a métodos situados en máquinas remotas a través de una red. Además, ORB maneja la transferencia de estructuras de datos de manera que sean compatibles entre los dos objetos. ORB es un componente fundamental de la arquitectura CORBA y su misión es facilitar la comunicación entre objetos. ORB también debe de facilitar diferentes transparencias, tales como de localización, implementación, comunicación y ejecución.

En CORBA [Coulouris et al., 2001], el servidor crea objetos remotos, hace referencias accesibles a objetos remotos y espera a que los clientes invoquen a estos objetos remotos o a sus métodos. Por su parte, el cliente obtiene una referencia de uno o más objetos remotos en el servidor e invoca a sus métodos.

La arquitectura CORBA (ver figura 3.13) está diseñada para dar soporte al rol de un intermediario de peticiones de objetos que permita que los clientes invoquen métodos en objetos remotos, donde tanto los clientes como los servidores pueden implementarse en diversos lenguajes de programación.

Figura 3.13. Principales componentes de la arquitectura CORBA [Coulouris et al., 2001]



La función de los principales componentes se describen a continuación [Coulouris et al., 2001]. El núcleo de ORB es un módulo de comunicación el cual permite una interfaz que incluye las operaciones de arranque y paro, las operaciones para la conversión entre referencias a objetos remotos y cadenas de texto, así como las operaciones para obtener listas de argumentos para llamadas que emplean invocación dinámica. El adaptador de objeto sirve como puente entre los objetos con interfaces IDL y las interfaces IDL, además de las interfaces del lenguaje de programación de las correspondientes clases sirvientes. Las clases esqueleto se generan en el lenguaje del servidor a través de un compilador de IDL. El esqueleto sirve para despachar las invocaciones a los métodos remotos, asimismo desempaqueta los argumentos desde los mensajes de petición y empaqueta las excepciones y los resultados en mensajes de respuesta. Los resguardo/proxy son los encargados de empaquetar los argumentos de los mensajes de invocación y desempaqueta las excepciones y los resultados de las respuestas. Cada repositorio de implementación activa los servidores registrados bajo demanda y localiza los servidores que están en ejecución en cada momento usando el nombre del adaptador de objeto. El repositorio de interfaz proporciona información sobre las interfaces IDL registradas a los clientes y a los servidores que lo requieran. La interfaz de invocación dinámica permite que los clientes puedan realizar invocaciones dinámicas sobre objetos remotos CORBA cuando no es práctico el uso de proxies. La interfaz de esqueleto dinámica permite que un objeto CORBA acepte invocaciones sobre una interfaz para la que no hay un esqueleto. El código delegado permite convertir a objetos CORBA aquellos fragmentos de código existente que fue diseñado sin prever los objetos distribuidos.

Ejercicios

1. Ilustra la arquitectura cliente-servidor para una comunicación de muchos a muchos.
    - Para ilustrar la arquitectura cliente-servidor para una comunicación de muchos a muchos, podemos imaginar un escenario en el que varios clientes necesitan acceder y compartir recursos entre sí a través de un servidor central. Cada cliente actúa como un nodo en la red y puede enviar solicitudes al servidor para acceder a los recursos compartidos. A su vez, el servidor gestiona estas solicitudes y coordina la comunicación entre los diferentes clientes, permitiendo así una comunicación de muchos a muchos.
2. Cita un ejemplo del uso de un proxy en una arquitectura distribuida.

- Un ejemplo del uso de un proxy en una arquitectura distribuida es el servidor proxy web. Este actúa como intermediario entre las solicitudes de recursos de los clientes y los servidores web. Cuando un cliente solicita un recurso a un servidor web, lo hace a través del servidor proxy, que luego reenvía la solicitud al servidor destino. El servidor proxy puede realizar diversas funciones, como caché de contenido, control de acceso, registro de tráfico y mejora del rendimiento.

3. Cita al menos tres desventajas del modelo cliente-servidor.
   - Tres desventajas del modelo cliente-servidor son: 1) Dependencia de un servidor central, lo que puede causar cuellos de botella y puntos únicos de falla. 2) Escalabilidad limitada, ya que el servidor puede volverse sobrecargado con un gran número de solicitudes de clientes. 3) Complejidad en el mantenimiento y la gestión del servidor y la infraestructura de red asociada.
4. Cita al menos tres características de la arquitectura peer-to-peer.
   - Tres características de la arquitectura peer-to-peer son: 1) Ausencia de un servidor centralizado, lo que permite una mayor escalabilidad y resiliencia frente a fallos. 2) Capacidad para compartir recursos directamente entre pares sin necesidad de intermediarios. 3) Flexibilidad para añadir nuevos nodos a la red sin afectar su funcionamiento general.
5. Explica cuál es la función del middleware en los sistemas distribuidos.
   - La función del middleware en los sistemas distribuidos es proporcionar una capa de abstracción que oculte la complejidad de la infraestructura subyacente y facilite la comunicación y la coordinación entre los diferentes componentes distribuidos. Esto se logra ofreciendo servicios para la distribución de datos y procesos a través de la red, así como gestionando la heterogeneidad de los sistemas distribuidos.
6. ¿Cómo está involucrado el uso del caché en los sistemas distribuidos?
   - El caché en los sistemas distribuidos se utiliza para almacenar copias de datos o recursos que se acceden con frecuencia, con el fin de reducir el tiempo de acceso y mejorar el rendimiento del sistema. Cuando un cliente solicita un recurso, el sistema primero verifica si está disponible en la caché local antes de buscarlo en el servidor remoto. Si el recurso está en caché, se entrega al cliente de manera más rápida, evitando así la necesidad de acceder al servidor remoto.
7. ¿Cuál es la diferencia entre una red superpuesta y una red física?
   - La diferencia entre una red superpuesta y una red física radica en su naturaleza y su función. Una red física se refiere a la infraestructura física de cables, routers, switches y otros dispositivos de red que proporcionan la conectividad física entre los nodos. Por otro lado, una red superpuesta es una abstracción lógica construida sobre la red física, donde los nodos pueden estar conectados de manera virtual sin importar su ubicación física. La red superpuesta permite establecer comunicación entre nodos que pueden estar dispersos geográficamente, sin depender de la topología física de la red subyacente.
8. ¿Qué beneficio aporta trabajar con CORBA en los sistemas distribuidos?
   - Trabajar con CORBA en los sistemas distribuidos aporta varios beneficios, como la capacidad de desarrollar aplicaciones distribuidas en entornos heterogéneos, la transparencia en la comunicación entre objetos remotos en diferentes sistemas, y la gestión de la complejidad asociada con el desarrollo de aplicaciones distribuidas. CORBA también proporciona un modelo de programación basado en principios orientados a objetos y soporta la interoperabilidad entre diferentes sistemas y lenguajes de programación.

9. Investiga las limitantes que tiene CORBA.
    - CORBA tiene limitaciones, como la complejidad de su implementación y configuración, la necesidad de definir interfaces IDL para cada objeto distribuido, y la dependencia de la infraestructura subyacente, lo que puede afectar la escalabilidad y el rendimiento del sistema. Además, CORBA puede enfrentar problemas de interoperabilidad entre diferentes implementaciones y versiones del middleware.
10. ¿Cuál es la diferencia entre una arquitectura grid y un clúster?
    - La principal diferencia entre una arquitectura grid y un clúster radica en su diseño y enfoque. Un clúster es un conjunto de computadoras que se comportan como una sola unidad mediante el uso de hardware común, generalmente con el mismo sistema operativo y configuración. En cambio, una arquitectura grid es una infraestructura de gestión de recursos distribuidos que se centra en el acceso coordinado a los recursos informáticos remotos, que pueden ser heterogéneos y dispersos geográficamente. Los clústeres son más adecuados para aplicaciones específicas y de alto rendimiento, mientras que las grids son más flexibles y escalables, permitiendo el acceso a recursos adicionales y el balanceo de carga.
11. ¿Cuál es la principal diferencia entre el paradigma cliente-servidor y el paradigma peer-to-peer?
    - La principal diferencia entre el paradigma cliente-servidor y el paradigma peer-to-peer es la centralización. En el modelo cliente-servidor, hay un servidor central que gestiona y distribuye recursos a los clientes, mientras que en el modelo peer-to-peer, no hay un servidor centralizado y los recursos son compartidos directamente entre los pares en la red. Esto hace que el modelo peer-to-peer sea más descentralizado y escalable, ya que no depende de un punto único de falla y permite una mayor flexibilidad en la comunicación y el intercambio de recursos.

Capítulo 4. Procesos y comunicación

Objetivo: Que el alumno comprenda los diferentes conceptos relacionados con los procesos y la comunicación distribuidos como la base para el desarrollo de aplicaciones distribuidas.

4.1 Introducción

La comunicación entre procesos es el núcleo de todos los sistemas distribuidos [Tanenbaum & Van Steen, 2008], por tal razón es importante entender la manera en que los procesos localizados en diferentes computadoras pueden intercambiar información. En los sistemas distribuidos tradicionalmente la comunicación está basada en el paso de mensaje. Esta técnica aporta sincronización entre procesos y permite la exclusión mutua, su principal característica es que no requiere memoria compartida, por lo que resulta ser muy importante en la programación de sistemas distribuidos. En este capítulo revisamos diferentes conceptos relacionados con los procesos y la comunicación en los sistemas distribuidos, como hilos, clientes, servidores, la llamada a un procedimiento remoto (RPC), el paradigma cliente-servidor, la comunicación en grupo y la interfaz de sockets.

4.2 Hilos

Los hilos se diferencian de los procesos en que los primeros comparten los mismos recursos del programa que las contiene, en tanto los procesos tienen de manera separada su código, así como sus datos. Se pueden identificar hilos de dos tipos de flujo:

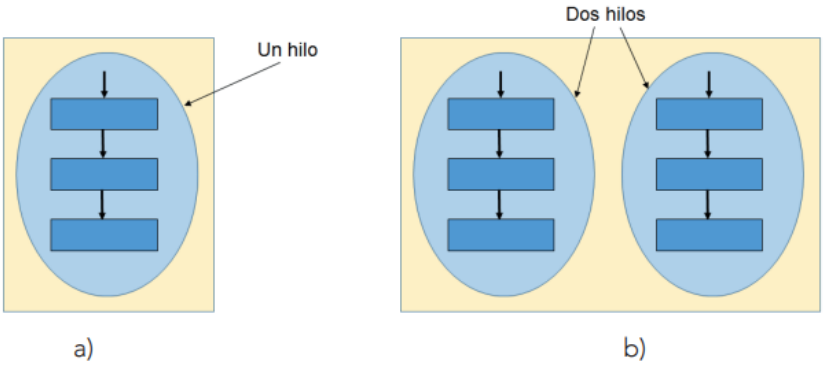
• Flujo único: En este caso, un programa utiliza únicamente un hilo para controlar su ejecución.
• Flujo múltiple: Son aquellos programas que utilizan varios contextos de ejecución para realizar su trabajo.

En un sistema multihilos, cada tarea se inicia y termina tan pronto como sea posible, esto facilita la entrada de datos en sistemas en tiempo real, especialmente si estos datos provienen de diferentes fuentes. En un programa multihilo se tiene el hilo principal del programa en ejecución, quien a su vez tiene otros hilos o tareas paralelas en ejecución. Un hilo se define como una secuencia única de control de flujo dentro de un programa, en un programa puede haber más de una secuencia de control o hilos. Un hilo es una parte del programa que se ejecuta independientemente del resto. El hilo es la unidad de código más pequeña que se pueda ejecutar en un entorno multitareas. El uso de hilos permite al programador escribir programas más eficientes, debido a que los hilos permiten optimizar recursos tan importantes como el mejor desempeño del CPU al minimizar sus tiempos de inactividad. El uso de hilos es muy valioso en entornos interactivos en red, ya que permiten sincronizar la diferencia entre la velocidad de transmisión de la red con las de procesamiento del CPU. La velocidad en el manejo del sistema de archivos para lectura y grabación es más lenta comparada con la velocidad de procesamiento de estos datos por el

CPU, en este caso el uso de hilos ayuda mucho. Una de las razones de importancia para el estudio de la programación multihilos es que permite acceder a los recursos de tiempo libre de la CPU mientras se realizan otras tareas. La figura ilustra esquemáticamente un programa de un hilo y un programa de dos hilos.

Figura 4.1. Comparación entre hilos:

a) Programa con un solo hilo, b) Programa multihilos



4.3 Cliente

En un sistema distribuido, el cliente es el elemento que solicita y usa el servicio que proporciona una funcionalidad específica o dato. El cliente tiene una postura proactiva, esto quiere decir que está trabajando en una tarea específica y cuando necesita cierto dato o una operación específica invoca al servidor para obtenerlo. Generalmente es por medio de la aplicación cliente que un usuario accede y mantiene un diálogo con el sistema. El usuario realiza el diálogo vía una interfaz gráfica de usuario. La operación del cliente consiste en arrancar, realizar su trabajo y terminar.

4.4 Servidores

El servidor es el elemento que proporciona la funcionalidad o servicio en un sistema distribuido. Este servicio puede consistir en compartir datos, informar sobre una solicitud, compartir recursos físicos, imprimir, etc. Generalmente se considera que un servidor tiene una posición reactiva en el sistema, ya que se encuentra inactivo hasta que recibe una petición de servicio.
Cuando recibe la petición, la procesa y envía la respuesta al solicitante, para después quedar nuevamente inactivo en espera de una nueva petición. En un sistema, el modo de ejecución de un servidor es continuo, ya que se inicializa al arrancar el sistema y está en operación hasta que el sistema se apaga. Un servidor tiene dos modos de arranque:

- Estático.
- Dinámico.

Un arranque estático sucede cuando el servidor se arranca como parte del arranque general del sistema donde se encuentra localizado. En contraste, un arranque dinámico sucede cuando un servidor es activado por un proceso del cliente que solicite sus servicios.
Los servidores forman parte importante del middleware básico de un sistema distribuido. Entre las características fundamentales de los servidores destaca que pueden ser reusables y relocalizarse. Los servidores pueden ser usados en diferentes tareas, entre las que se pueden destacar las siguientes:

• Servidores de datos.
• Servidores de archivos.
• Servidores de impresión.
• Servidores de correo.
• Servidores de programas.
• Servidores de bases de datos.
• Servidores de fecha y hora.
• Servidores de multimedia.
• Servidores de transacciones.
• Servidores web.

4.5 Comunicación entre procesos

La comunicación entre procesos es un factor clave para construir sistemas distribuidos, los paradigmas de comunicación más usados en sistemas distribuidos son:
• Cliente - servidor.
• Llamada a un procedimiento remoto (RPC).
• Comunicación en grupo.

Los conceptos fundamentales que deben ser considerados para la comunicación son:

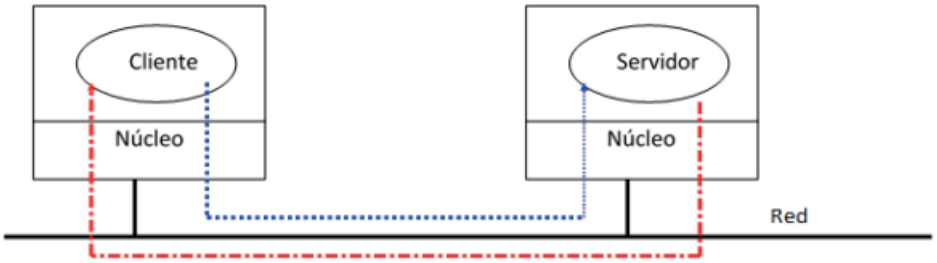
• Los datos tienen que ser aplanados antes de ser enviados.
• Los datos tienen que ser representados de la misma manera en la fuente y destino.
• Los datos tienen que empaquetarse para ser enviados.
• Usar operaciones de send para enviar y receive para recibir.
• Especificar la comunicación, ya sea en modo bloqueante o no bloqueante.
• Abstracción del mecanismo de paso de mensaje.
• La confiabilidad de la comunicación. Por ejemplo, usar TCP en lugar de UDP.

4.5.1 Modelo cliente - servidor (C-S)

El término cliente - servidor (C-S) hace referencia a la comunicación en la que participan dos aplicaciones. Es decir que está basado en la comunicación de uno a uno.
La aplicación que inicia la comunicación enviando una petición y esperando una respuesta se llama cliente. Mientras los servidores esperan pasivos, aceptan peticiones recibidas a través de la red, realizan el trabajo y regresan el resultado o un código de error porque no

se generó la petición. Una máquina puede ejecutar un proceso o varios procesos cliente. La transferencia de mensaje en el modelo C-S se ejecuta en el núcleo. Una operación general del funcionamiento del cliente - servidor se muestra en la figura 4.2, donde un servidor espera una petición sobre un puerto bien conocido que ha sido reservado para cierto servicio. Un cliente reserva un puerto arbitrario y no usado para poder comunicarse.

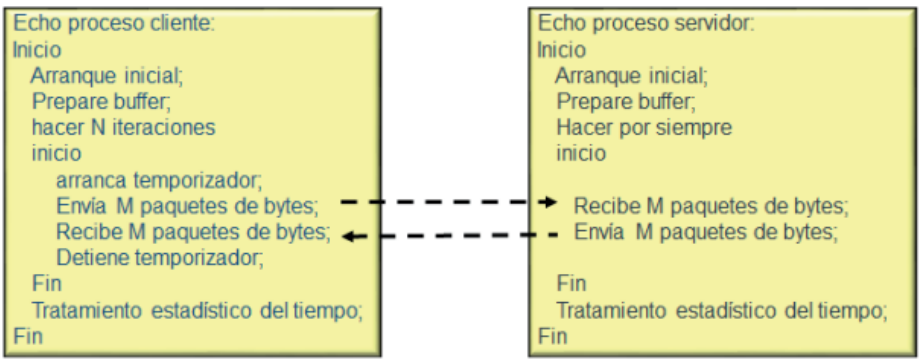
Figura 4.2. Modelo cliente - servidor



Características del software cliente - servidor

El software cliente es un programa de aplicación que se vuelve cliente temporal cuando se necesita acceso remoto. Este programa llama directamente al usuario y se ejecuta sólo durante una sesión. Se ejecuta localmente en la computadora del usuario y se encarga de iniciar el contacto con el servidor. Este programa puede acceder a varios servicios y no necesita hardware especial ni un complejo sistema operativo. Por su parte, el software servidor es un programa especial de propósitos dedicados a ofrecer un servicio pero que puede manejar varios clientes remotos al mismo tiempo. Este programa se inicia automáticamente al arranque del sistema y continuará ejecutándose en varias sesiones. El servidor espera pasivamente el contacto de los clientes remotos. Por lo general, el software servidor necesita ser instalado en un sistema operativo más robusto y contar con mucha capacidad de recursos de hardware. En la figura 4.3 se muestra un seudocódigo para un eco (echo) para un cliente - servidor (C-S).

Figura 4.3. Seudocódigo para un eco entre cliente y servidor [Lin, Hsieh, Du, Thomas & McDonald, 1995]



La comunicación cliente - servidor puede ser implementada a través de librerías PVM (Parallel Virtual Machine) o sockets que permiten implementar comunicaciones bloqueantes y no bloqueantes, además de que también pueden usar TCP para tener comunicaciones confiables.

Deficiencias del modelo cliente - servidor

• El paradigma de comunicación es la entrada/salida (E/S), ya que estos no son fundamentales en el cómputo centralizado.
• No permite la transparencia requerida para un ambiente distribuido.
• El programador debe de atender la transferencia de mensajes o las E/S, tanto del lado del cliente como del lado del servidor.

4.5.2 Llamada de procedimiento remoto (RPC)

La llamada de procedimiento remoto, mejor conocido como RPC, es una variante del paradigma cliente - servidor y es una vía para implementar en la realidad este paradigma. En el RPC, un programa llama a un procedimiento localizado en otra máquina. El programador no se preocupa por las transferencias de mensajes o de las E/S. La idea de RPC es que una llamada a un procedimiento remoto se parezca lo más posible a una llamada local, esto le permite una mayor transparencia. Para obtener dicha transparencia, el RPC usa un resguardo de cliente, que se encarga de empacar los parámetros en un mensaje y le solicita al núcleo que envíe el mensaje al servidor, posteriormente se bloquea hasta que regrese la respuesta.

Algunos puntos problemáticos del RPC son:

• Que se deben de usar espacios de direcciones distintos, debido a que se ejecutan en computadoras diferentes.
• Como las computadoras pueden no ser idénticas, la transferencia de parámetros y resultados puede complicarse.
• Ambas computadoras pueden descomponerse.

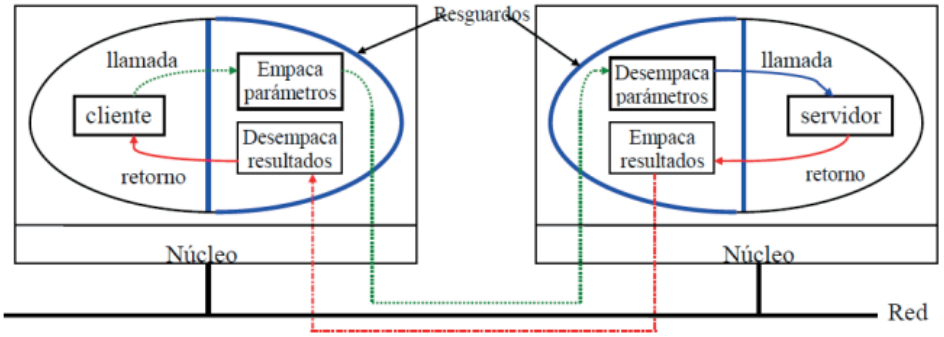
Modo de operación del RPC

En la figura 4.4 se muestra una llamada a un procedimiento remoto, el cual se realiza considerando los siguientes pasos:

1. El procedimiento cliente llama al resguardo del cliente de la manera usual.
2. El resguardo del cliente construye un mensaje y realiza un señalamiento al núcleo.
3. El núcleo envía el mensaje al núcleo remoto.
4. El núcleo remoto proporciona el mensaje al resguardo del servidor.
5. El resguardo del servidor desempaca los parámetros y llama al servidor.
6. El servidor realiza el trabajo y regresa el resultado al resguardo.
7. El resguardo del servidor empaca el resultado en un mensaje y hace un señalamiento mediante el núcleo.

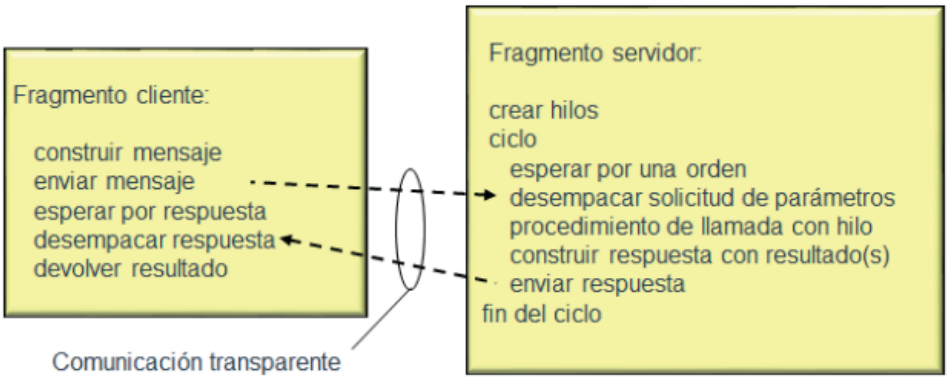
8. El núcleo remoto envía el mensaje al núcleo del cliente.
9. El núcleo del cliente da el mensaje al resguardo del cliente.
10. El resguardo desempaca el resultado y lo entrega al cliente.

Figura 4.4. Modo de operación de un RPC entre un proceso cliente y uno servidor



En el seudocódigo de la figura 4.5 se indica un ejemplo de fragmentos de un RPC.

Figura 4.5. Fragmentos de cliente - servidor en RPC

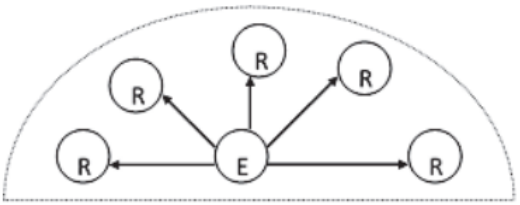


4.5.3 Comunicación en grupo

En un sistema distribuido puede haber comunicación entre procesos de uno a muchos o de uno a muchos, como se indica en la figura 4.6. Los grupos son dinámicos, lo cual implica que se pueden crear nuevos grupos y destruir grupos anteriores. Las técnicas para implantar la comunicación en grupos pueden ser:

- Transmisión de multidifusión (multicast)
- Transmisión completa
- Transmisión punto a punto

Figura 4.6. La comunicación en grupo se da entre un emisor (E) y varios receptores (R)



Con respecto a los aspectos de diseño de los grupos, se tienen las siguientes opciones [Tanenbaum, 1996]:

• Grupos cerrados: Por ejemplo, el procesamiento paralelo.
• Grupos abiertos: Por ejemplo, el soporte de servidores redundantes.
• Grupo de compañeros: Permite que las decisiones se tomen de manera colectiva.
• Ventaja: el grupo de compañeros es simétrico y no tiene punto de falla.
• Desventaja: la toma de decisiones por votación es difícil, crea retraso y es costosa.
• Grupo jerárquico: Existe un coordinador y varios trabajadores.
• Membresía de grupo:
• Permite crear, eliminar grupos, agregar o eliminar procesos de grupos.
• Utiliza técnicas como la del servidor del grupo o membresía distribuida.
• Existen problemas de detección cuando un miembro ha fallado.
• Direccionamiento al grupo: Los grupos deben de poder direccionarse.
Estas pueden ser:
• Dirección única grupal
• Dirección de cada miembro del grupo
• Direccionamiento de predicados
• Primitivas: Las primitivas de comunicación son:
• send y receive, de la misma forma que en una comunicación puntual
• group_send y grop_receive para comunicación en grupo
• Atomicidad: Se refiere a que cuando se envía un mensaje a un grupo, este debe de llegarles a todos los miembros o a ninguno, así como garantizar la consistencia.
• Ordenamiento de mensajes: Está conjuntada con la atomicidad y permite que la comunicación en grupo sea fácil de comprender y utilizar. Los criterios son:
• Ordenamiento con respecto al tiempo global
• Ordenamiento consistente
• Ordenamiento con respecto a grupos traslapados
• Escalabilidad: Permite que el grupo continúe funcionando, aun cuando se agreguen nuevos miembros.

4.6 Interfaz de programación de aplicaciones (API)

Las aplicaciones clientes - servidor se valen de protocolos de transporte para comunicarse. Las aplicaciones deben de especificar los detalles sobre el tipo de servicio (cliente o servidor), los datos a transmitir y el receptor donde situar los datos. La interfaz entre un

programa de aplicación y los protocolos de comunicación de un sistema operativo se llama interfaz de programación de aplicaciones (API).

Un API define un conjunto de procedimientos que puede efectuar una aplicación al interactuar con el protocolo. Por lo general, un API tiene procedimientos para cada operación básica. Los programadores pueden escoger una gran variedad de APIs para sistemas distribuidos. Algunas de estas son BSD socket, Sun's RPC/XDR, librería de paso de mensajes PVM y Windows Sockets. La API de socket, que tuvo sus orígenes en el UNIX BSD, se convirtió en una norma de facto en la industria y muchos sistemas operativos la han adoptado, tanto para computadoras basadas en Widows (winsock) como para algunos sistemas UNIX.

4.6.1 La interfaz de socket

Un socket es un punto de referencia hacia donde los mensajes pueden ser enviados, o de donde pueden ser recibidos. Al llamar la aplicación a un procedimiento de socket, el control pasa a una rutina de la biblioteca de sockets que realiza las llamadas al sistema operativo para implementar la función de socket. UNIX BSD y los sistemas derivados contienen una biblioteca de sockets, la cual puede ofrecer a las aplicaciones una API de socket en un sistema de cómputo que no ofrece sockets originales.
Los sockets emplean varios conceptos de otras partes del sistema, pero en particular están integrados con la E/S, por lo que una aplicación se comunica con un socket de la misma forma en que se transfiere un archivo. Cuando una aplicación crea un socket, recibe un descriptor para hacer referencia a este. Si un sistema usa el mismo espacio de descriptor para los sockets y otras E/S, es posible emplear la misma aplicación para transferir datos localmente como para su uso en red. La ventaja del método de socket es que la mayor parte de las funciones tienen tres o menos parámetros, sin embargo, se debe llamar a varias funciones a este método.

4.6.2 Funciones de la API de sockets

Las funciones relacionadas a la API sockets que permiten establecer una comunicación entre dos computadoras son:

socket( )
Esta rutina se usa para crear un socket y regresa un descriptor correspondiente a este socket. Este descriptor es usado en el lado del cliente y en el lado del servidor de su aplicación. Desde el punto de vista de la aplicación, el descriptor de archivo es el final de un canal de comunicación. La rutina retorna -1 si ocurre un error.

close( )
Indica al sistema que el uso de un socket debe de ser finalizado. Si se usa un protocolo TCP (orientado a conexión), close termina la conexión antes de cerrarlo. Cuando el socket se cierra, se libera al descriptor, por lo que la aplicación ya no transmite datos y el protocolo de transportación ya no acepta mensajes de entradas para el socket.

bind( )
Suministra un número a una dirección local a asociar con el socket, ya que cuando un socket es creado no cuenta con dirección alguna.

listen( )
Esta rutina prepara un socket para aceptar conexiones y solo puede ser usada en sockets que utilizan un canal de comunicación virtual. Esta rutina se deberá usar del lado del servidor de la aplicación antes de que se pueda aceptar alguna solicitud de conexión del lado del cliente. El servidor encola las solicitudes de los clientes conforme estas llegan. La cola de solicitudes permite que el sistema detenga las solicitudes nuevas mientras que el servidor se encarga de las actuales.

accept( )
Esta rutina es usada del lado del servidor de la aplicación para permitir aceptar las conexiones de los programas cliente. Después de configurar una cola de datos, el servidor llama accept, cesa su actividad y espera una solicitud de conexión de un programa cliente. Esta rutina solo es válida en proveedores de transporte de circuito virtual. Cuando llega una solicitud al socket especificado accept( ) llena la estructura de la dirección, con la dirección del cliente que solicita la conexión y establece la longitud de la dirección, accept( ) crea un socket nuevo para la conexión y regresa su descriptor al que lo invoca, este nuevo socket es usado por el servidor para comunicarse con el cliente y al terminar se cierra. El socket original es usado por el servidor para aceptar la siguiente conexión del cliente.

connect( )
Esta rutina permite establecer una conexión a otro socket. Se utiliza del lado del cliente de la aplicación permitiendo que un protocolo TCP inicie una conexión en la capa de transporte para el servidor especificado. Cuando se utiliza para protocolos sin conexión, esta rutina registra la dirección del servidor en el socket, esto permite que el cliente transmita varios mensajes al mismo servidor. Usualmente el lado cliente de la aplicación enlaza a una dirección antes de usar esta rutina, sin embargo, esto no es requerido.

send( )
Esta rutina es utilizada para enviar datos sobre un canal de comunicación tanto del lado del cliente como del lado servidor de la aplicación. Se usa para sockets orientados a conexión, sin embargo, podría utilizarse para datagramas pero haciendo uso de connect( ) para establecer la dirección del socket.

sendto( )
Permite que el cliente o servidor transmita mensajes usando un socket sin conexión (usando datagramas). Es exactamente similar a send( ) solo que se deberán especificar la dirección destino del socket al cual se quiere enviar el dato. Se puede usar en sockets orientados a conexión pero el sistema ignorará la dirección destino indicada en sendto( ).

recv( )
Esta rutina lee datos desde un socket conectado y es usado tanto en el lado del cliente como del lado del servidor de la aplicación.

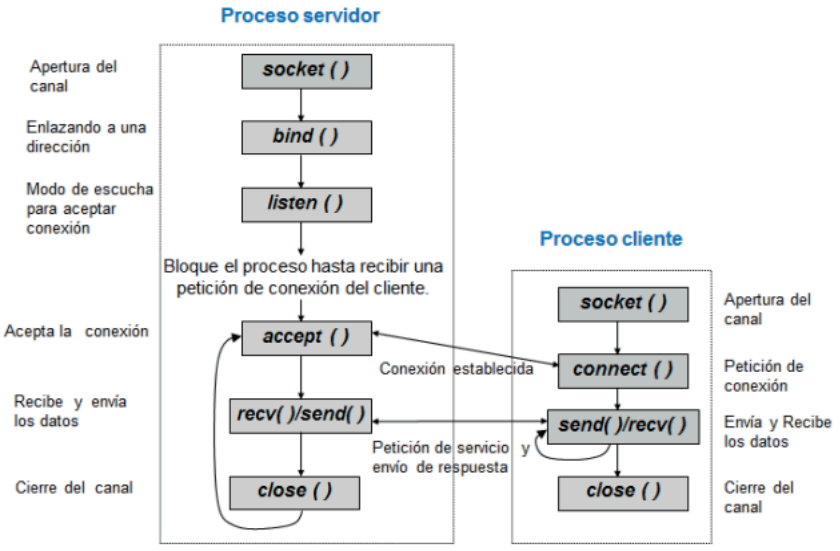
recvfrom( )
Esta rutina lee datos desde un socket sin conexión. En este caso, el sistema regresa la dirección del transmisor con los mensajes de entrada y permite registrar la dirección del socket transmisor en la misma forma que espera sendto( ), por lo que la aplicación usa la dirección registrada como destino de la respuesta.

4.6.3 Ejemplo de cliente servidor usando socket

4.6.3.1 Comunicación orientada a conexión (TCP)

Un servicio orientado a conexión requiere que dos aplicaciones establezcan una conexión de transportación antes de comenzar el envío de datos. Para establecer la comunicación, ambas aplicaciones primero interactúan localmente con protocolo de transporte y después estos protocolos intercambian mensajes por la red. Una vez que ambos extremos están de acuerdo y se haya establecido la conexión, las aplicaciones podrán enviar datos. La secuencia de llamadas para un escenario orientado a conexión se indica en la figura 4.7.

Figura 4.7. Uso del API socket para una comunicación orientada a conexión [Jamsa & Cope, 1996]

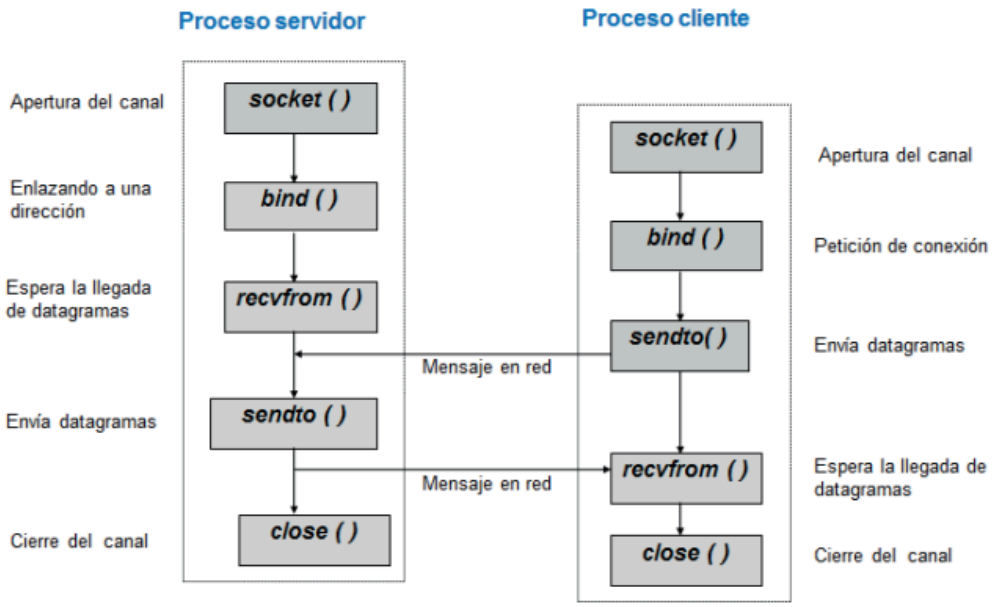


En la figura 4.7 se observa que, después de llamar a listen( ), el servidor se pone en modo pasivo hasta que recibe una solicitud del cliente. Cuando la solicitud de servicio del cliente llega al socket monitoreado por la función accept( ), se crea de modo automático un socket nuevo, que será conectado inmediatamente con el proceso cliente. Este socket se llama de servicio y se cierra al terminar la transferencia de datos. El socket original que recibió la solicitud de servicio permanece abierto en estado de escucha para seguir aceptando nuevas solicitudes.

4.6.3.2 Comunicación sin conexión o servicio de datagrama

El servicio de datagrama (ver figura 4.8) [Jamsa & Cope, 1996] es similar a enviar una carta postal. El lado cliente actúa como la persona que envía la carta y el lado servidor como la persona que lo recibe. El servidor usa las llamadas a socket( ) y bind( ) para crear y unir un socket. Como el socket es sin conexión, se deberán usar las llamadas a recvfrom( ) y sendto( ). Se llama a la función bind( ) pero no a connect( ), dejando esta última como opcional. La dirección destino se especifica en la función sendto( ). La función recvfrom( ) no espera conexión sino que responde a cualquier dato que llegue al puerto que tiene enlazado.

Figura 4.8. Comunicación para aplicaciones que usan datagramas [Jamsa & Cope, 1996]



Ejercicios

1. Explica las tres cosas que debe garantizar un MUTEX.
   - Un MUTEX, o semáforo binario, debe garantizar tres cosas principales: exclusión mutua, que significa que solo un hilo puede acceder a un recurso compartido a la vez; progreso limitado, para evitar que los hilos se queden bloqueados indefinidamente; y espera ocupada, lo que significa que un hilo que no puede acceder al recurso en un momento dado no debe consumir recursos de CPU mientras espera.
2. Dadas las variables X y Z, con hilo H1 fijando ambas variables a 0 mientras hilo H2 fija a ambas variables en 1, escribe el seudocódigo de un MUTEX que garantice la consistencia de ambas variables.

- El seudocódigo para un MUTEX que garantiza la consistencia de ambas variables X y Z en un contexto donde el hilo H1 fija ambas variables a 0 y el hilo H2 las fija a 1 podría ser:

```
inicio
   variable mutex inicializada a 1
   H1:
      while true do
         P(mutex)
         X = 0
         Z = 0
         V(mutex)
   H2:
      while true do
         P(mutex)
         X = 1
         Z = 1
         V(mutex)
fin
```

- Este código asegura que solo un hilo puede acceder a la sección crítica (la modificación de las variables X y Z) a la vez, garantizando así la consistencia de ambas variables.

3. Multicast es una tecnología para comunicación en grupo, explica su funcionamiento.
   - Multicast es una tecnología para comunicación en grupo que permite enviar un mensaje desde un solo emisor a múltiples receptores simultáneamente. Funciona enviando un único paquete de datos desde el emisor a una dirección de grupo, y luego el enrutador de red se encarga de replicar este paquete y enviarlo a todos los receptores que se han unido al grupo.
4. Cita al menos tres funciones relacionadas con los hilos y el uso de cada una de ellas.
   - Funciones relacionadas con los hilos incluyen:
     - pthread_create(): para crear un nuevo hilo.
     - pthread_join(): para esperar a que un hilo termine su ejecución.
     - pthread_mutex_lock(): para adquirir un mutex y bloquear el acceso a una sección crítica.
5. Cita al menos tres desventajas del modelo cliente - servidor.
   - Algunas desventajas del modelo cliente-servidor incluyen:
     - Paradigma de E/S: el modelo está basado en E/S, lo que puede no ser óptimo en entornos distribuidos.
     - Falta de transparencia: no ofrece la transparencia necesaria para un entorno distribuido.
     - Complejidad de programación: los programadores deben ocuparse de la transferencia de mensajes y E/S tanto en el lado del cliente como del servidor.
6. Explica la diferencia entre el RPC y el modelo cliente - servidor.
   - La diferencia principal entre el RPC y el modelo cliente-servidor radica en el nivel de abstracción proporcionado al programador. En el RPC, el programador llama a un procedimiento remoto como si fuera local, lo que oculta la complejidad de la comunicación de red. Mientras que en el modelo cliente-servidor, el programador

debe manejar directamente la transferencia de mensajes y la E/S tanto en el lado del cliente como del servidor.

7. Explica la diferencia entre envío de datagrama y flujo de datos.
    - La diferencia entre el envío de datagramas y el flujo de datos radica en el modo de comunicación. El envío de datagramas se utiliza en comunicaciones sin conexión, donde los mensajes se envían como paquetes independientes que pueden llegar en cualquier orden y pueden perderse. Mientras que el flujo de datos se utiliza en comunicaciones orientadas a conexión, donde se establece una conexión antes de enviar datos, garantizando así la entrega secuencial y fiable de los mensajes.
8. ¿Cuándo es importante usar una comunicación orientada a conexión?
    - Es importante usar una comunicación orientada a conexión cuando se necesita una transferencia fiable y secuencial de datos, como en la transmisión de archivos o en la comunicación entre aplicaciones que requieren integridad de datos.
9. ¿Cuándo es importante usar una comunicación sin conexión?
    - Es importante usar una comunicación sin conexión cuando la entrega rápida de datos es prioritaria y la integridad de los datos no es crítica, como en la transmisión de datos en tiempo real o en aplicaciones donde la pérdida ocasional de datos no es un problema.
10. ¿Cuál es la diferencia entre un servidor de transacciones y un servidor de archivos?
    - La diferencia principal entre un servidor de transacciones y un servidor de archivos radica en la naturaleza de los servicios que proporcionan. Un servidor de transacciones se utiliza para coordinar y gestionar transacciones en sistemas distribuidos, asegurando la consistencia de los datos, mientras que un servidor de archivos proporciona acceso a archivos y directorios almacenados en un sistema de archivos remoto.
11. ¿Cuál es la diferencia entre una función send( ) y sendto( )?
    - La diferencia entre send() y sendto() radica en el contexto de su uso. send() se utiliza en comunicaciones orientadas a conexión para enviar datos a través de un socket conectado, mientras que sendto() se utiliza en comunicaciones sin conexión para enviar datos a través de un socket sin necesidad de una conexión previa.