
	<p>Sistemas Operativos</p>	
---	-----------------------------------	---

Reporte de Exposición

<p>Arquitectura de Servidores</p>
--

<p>Nombre</p>	
<p>Pérez Saldaña Luis Mario</p>	
<p>Prof: Gunnar Eyal Wolf Iszaevich Ciclo: 2021-2</p>	<p>Grupo: 6</p>

Índice

Conceptos Introductorios	3
¿Qué es un servidor?	3
¿Qué es un puerto de red?	3
¿Qué es un socket de red?	4
Arquitecturas de Servidor Basadas en Hilos	5
Arquitecturas Multiproceso	6
Ejemplos	6
Arquitecturas Multihilo	6
Ejemplos	7
Arquitecturas de Servidor basadas en eventos	7
Arquitectura Hilo Único	7
Patrones de Diseño	8
Reactor	8
Comparación de Arquitecturas de Servidor	8
Efectos sobre la Mantenibilidad de Código	8
Efectos sobre Recursos	9
Efectos sobre Rendimiento	9

Conceptos Introductorios

¿Qué es un servidor?

En software, un servidor es un programa de computadora que realiza algunas tareas en nombre de otra pieza de software (cliente). Dado que no hay forma de que el servidor prediga cuando puede ser solicitado por una computadora cliente, un servidor normalmente se ejecuta en todo momento. Sin embargo, también existen algunas aplicaciones de servidor que se inician automáticamente cuando se recibe una solicitud de un cliente y luego se detienen cuando se ha satisfecho la solicitud.

En esta definición, es importante la distinción entre los servidores de tipo hardware y de tipo software. Un servidor también puede definirse como una computadora capaz de esperar por cualquier solicitud de un cliente y actuar sobre esa solicitud.. El sistema operativo (SO) del servidor está diseñado para respaldar operaciones basadas de red, brindar servicios, y procesar las cargas de trabajo entrantes y salientes. Prácticamente cualquier computadora con acceso a la red puede ser un servidor. Una computadora con "hardware de clase de servidor" implica que está especializada para ejecutar servidores en ella.

Los servidores más comunes son servidores web, servidores de bases de datos, servidores de archivos, servidores de correo, servidores de juegos y servidores FTP.

¿Qué es un puerto de red?

A nivel de software, dentro del sistema operativo, un puerto es una construcción lógica que identifica a un proceso específico en una computadora. Un puerto se identifica con un número, el número de puerto. En el modelo OSI (en inglés, Open Systems Interconnection), los puertos son construcciones a nivel de la capa de transporte; es decir, son abstracciones al mismo nivel de los protocolos, que se encargan que los datos sean entregados en su destino sin errores y en el mismo orden en que se transmitieron. Los protocolos de transporte más comunes que utilizan números de puerto son el Protocolo de control de transmisión (TCP) y el Protocolo de datagramas de usuario (UDP).

En el protocolo TCP, los paquetes contienen dos partes: el encabezado TCP y el cuerpo TCP. El encabezado TCP contiene información sobre los puertos de red. Específicamente, incluye el puerto de origen TCP, que especifica de qué proceso de cliente o servidor proviene el paquete en la máquina de origen. También incluye el puerto de destino TCP, similar al puerto de origen TCP.

A nivel sistema operativo, un proceso se asocia a un número de puerto y una dirección IP a través de un "socket" de red, que es un tipo de descriptor de archivo. Esto se conoce

como “binding”. En otras palabras, en una computadora con IP dada, el sistema operativo asocia lógicamente procesos y números de puertos, como si de una tabla se tratase. El sistema operativo tiene la tarea de transmitir datos salientes desde todos los puertos a la red, así como reenviar los paquetes de red que se reciben a los procesos correctos según el número de puerto del paquete. Es muy común que sólo 1 proceso puede vincularse a una combinación de puerto y dirección IP específica.

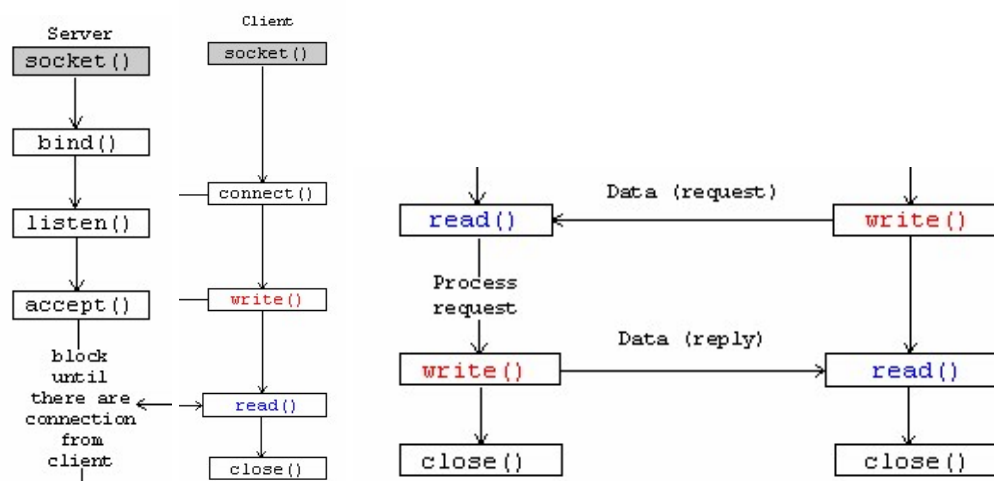
Los puertos de red numerados del 0 al 1023 en los protocolos TCP y UDP están reservados a tipos de aplicaciones específicas. Por ejemplo, el puerto 80 está reservado para aplicaciones web. IANA (Internet Assigned Numbers Authority) es la entidad encargada de dicha asignación.

¿Qué es un socket de red?

Según el contexto, “socket de red” puede representar conceptos diferentes. Primero, para explicar más claramente, se hará una distinción algo artificiosa entre un socket de “cliente”, un endpoint de una comunicación, y un socket de “servidor”.

Inicialmente, un servidor web crea un “socket servidor”. El socket puede hacer un enlazado (bind) entre el proceso actual, y un hostname con un puerto. El socket ahora es capaz de escuchar (listen) desde algún canal de entrada a las solicitudes que se le hagan. Ahora que se tiene un socket servidor escuchando en un puerto, ya es posible entrar al bucle principal del servidor, donde se aceptan (accept) las solicitudes recibidas de un cliente.

Un socket de cliente (en el cliente) se conecta (connect) inicialmente al servidor utilizando la dirección IP y el puerto al que se enlazó el socket de servidor. Cuando connect termina, el socket puede ser usado en una petición. El mismo socket leerá la respuesta y luego será destruido.



Cabe mencionar que el “socket cliente” en cliente y el “socket cliente” del servidor son estructuras análogas. Ambas tienen estructuras parecidas, y para el programador las interfaces son muy similares.

A nivel sistema operativo los “sockets de cliente” se manejan también muy similarmente a los archivos; cada socket también se identifica con un descriptor de socket.

Los sockets se implementan desde el sistema operativo en la llamada “pila de protocolos”, que es un conjunto de servicios que permiten que los procesos se comuniquen a través de una red. El sistema operativo reenvía la carga útil de los paquetes IP entrantes a la aplicación correspondiente extrayendo la información de la dirección del socket de los encabezados de protocolo de transporte y IP y eliminando los encabezados de los datos de la aplicación.

Arquitecturas de Servidor Basadas en Hilos

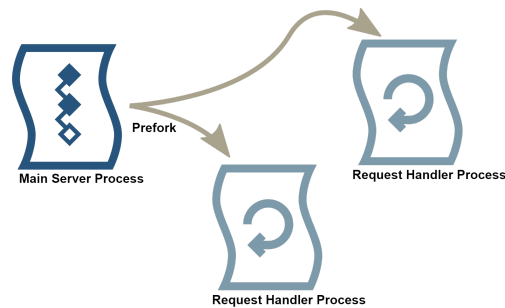
Una arquitectura basada en hilos asocia cada solicitud al servidor con un hilo (o proceso) dedicado. La concurrencia se maneja empleando múltiples hilos (o procesos). En esta arquitectura, se trata con operaciones de E/S de forma síncrona y bloqueante en cada hilo (o proceso). De esta forma, la programación para servidores no muy complejos se simplifica y el código resultante es más sencillo de seguir de forma secuencial. Sin embargo, para servidores más complejos es preciso controlar los problemas que surgen de manejar entornos multitarea, al diseñar robustamente la aplicación para evitar problemas comunes en entornos concurrentes, como los bloqueos mutuos ó las condiciones de carrera.

Las arquitecturas multiproceso y multihilo siguen el mismo principio: tener una actividad dedicada a procesar una solicitud hacia el servidor.

Arquitecturas Multiproceso

En una arquitectura multiproceso, un proceso dedicado toma el control sobre el procesamiento de una petición al servidor. Generalmente, existe un “socket de servidor” único, que es seguro en este entorno de concurrencia.

Como los procesos son estructuras costosas, los servidores a menudo emplean una estrategia llamada *preforking*. Cuando se utiliza el *preforking*, el proceso principal del servidor lanza varios procesos de forma preventiva al inicio de la ejecución del programa, y que serán los encargados de responder a las solicitudes. Con dicha estrategia, se dispone de un “pool” de procesos, donde uno se bloqueará para una conexión entrante, y al término del procesamiento de la solicitud, podrá ser reutilizado para la siguiente conexión.



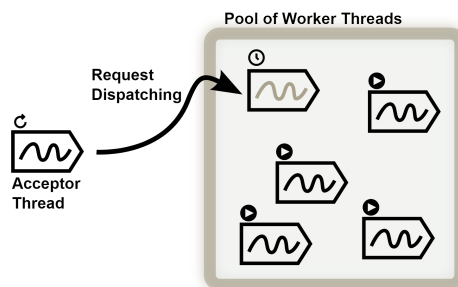
Ejemplos

El servidor web Apache proporciona un robusto módulo de multiproceso que se basa en preforking, llamado Apache MPM Prefork. Sigue siendo el módulo multiproceso predeterminado para las configuraciones de Apache basadas en UNIX.

Arquitecturas Multihilo

Existe una alternativa de arquitecturas de servidor que reemplaza los procesos pesados con *procesos ligeros*. Esto es, emplean un modelo de hilo por conexión. Aunque sigue los mismos principios, el enfoque de subprocesos múltiples tiene varias diferencias importantes. Por ejemplo, varios subprocesos comparten el mismo espacio de direcciones y, por lo tanto, comparten variables y estados globales. Los hilos son generalmente un reemplazo más eficiente cuando se asignan conexiones a actividades.

En la práctica, es una arquitectura común colocar un hilo *despachador* frente a un pool de hilos para el manejo de la conexión. Una vez establecida, la conexión se pasa a una cola de conexiones entrantes. Los hilos del pool disponibles toman conexiones de la cola, procesan las solicitudes y una vez terminados esperan nuevamente conexiones en la cola. Si se observa, es muy similar a la arquitectura multiproceso con preforking.



Ejemplos

El trabajador Apache-MPM en realidad es un módulo multiproceso para el servidor web Apache que combina procesos e hilos. El módulo genera varios procesos y cada proceso,

a su vez, administra su propio grupo de hilos. Es decir, es una arquitectura híbrida basada en hilos.

Arquitecturas de Servidor basadas en eventos

La arquitectura de servidor basada en eventos surge como alternativa a la naturaleza bloqueante y síncrona de la arquitectura basada en hilos. Un modelo común está orientado al manejo de eventos. Un evento es una acción u ocurrencia reconocida, que a menudo se va a tratar de forma asíncrona. Por ejemplo, al recibir una solicitud en un servidor, un evento podría ser la recepción de un chunk de bits desde un canal de entrada de red.

Los eventos que se van originando a lo largo de la ejecución del programa se colocan en una cola y un hilo ejecuta un llamado event-loop: retira los eventos de la cola, procesa el evento y toma el siguiente evento en caso de haberlo, o espera a que se envíen nuevos eventos. Previamente, al event-loop se le tuvo que señalar la forma de manejar a dichos eventos para que pueda procesarlos. Esto último es tarea del programador.

Arquitectura Hilo Único

Un modelo común es el mapeo de un solo hilo a múltiples conexiones. Luego, el hilo maneja todos los eventos que ocurren de las operaciones de E/S de estas conexiones y solicitudes. El event-loop se ejecuta sobre este hilo único.

Patrones de Diseño

Reactor

Al principio, una aplicación que sigue este patrón se “suscribe” a un conjunto de eventos que le interesan. Para cada evento de recurso que le interesa a la aplicación, se debe proporcionar una forma de procesar dicho evento: un callback (función), por ejemplo. Siempre que el event-loop recibe un evento, notifica a un despachador y espera el próximo evento. El despachador procesa el evento activando el(los) callback(s) proporcionado por el programador.

El patrón de reactor está destinado al manejo de E/S asíncrono y sin bloqueo.

Comparación de Arquitecturas de Servidor

El objetivo principal de las arquitecturas de servidor es proporcionar escalabilidad. Esto significa que los servidores web deben enfrentar los siguientes desafíos:

- Alta concurrencia: los servidores deberían poder manejar incluso miles de conexiones al mismo tiempo, idealmente sin sobrecarga de los recursos con los que se dispone.
- Alto rendimiento: los servidores deben ser capaces de proveer un tiempo de respuesta razonable a cada una de las peticiones que recibe, y soportar altas cantidades de tráfico de entrada y salida.
- Alta mantenibilidad: la base de código sobre la que se soporta un servidor debe ser mantenible a largo plazo, a medida que aumenta su complejidad.

La elección de una arquitectura es altamente influyente sobre los resultados que entregará un servidor.

Efectos sobre la Mantenibilidad de Código

Bajo ciertas condiciones, la arquitectura basada en hilos es conceptualmente más fácil de seguir: la programación con hilos y procesos se basa esencialmente en expresar un flujo de control lineal, como cualquier otro programa de ejecución secuencial. Los hilos son entidades bien conocidas y comprendidas de los sistemas operativos y son primitivas de propósito general para cualquier tipo de paralelismo.

Sin embargo, esto es necesario matizarlo. Es en realidad muy difícil escribir código concurrente correcto usando hilos. Es posible argumentar que una implementación con una correcta coordinación y sincronización es difícil de lograr incluso para los programadores experimentados.

Por otro lado, en una arquitectura basada en eventos podría argumentarse que es fácil confundir el flujo de control, y exige que el programador escriba más código. A veces también puede dar la impresión que se obtiene un código muy fragmentado. Otro argumento en contra de los sistemas basados en eventos es que la mantenibilidad de la programación para tales sistemas rápidamente se degrada hacia un infierno de “callbacks”, consecuencia del manejo asíncrono de eventos. Esto es, según algunos programadores, significativamente más complejo en comparación con la semántica simple de los hilos.

Efectos sobre Recursos

Una característica a considerar de la arquitectura basada en hilos es su alta huella de memoria en entornos de alta concurrencia. Bajo carga considerable, un servidor puede consumir grandes cantidades de recursos dado que se dedica un hilo (o peor, proceso) a cada petición recibida. Bajo cambios de contexto frecuentes, la situación se agrava pues se pierde tiempo considerable en tareas burocráticas del sistema operativo. Además, también es posible que con los cambios de contexto existan condiciones de obsolescencia del caché en los procesadores aún existiendo afinidad de procesos. Todos

estos factores limitan la escalabilidad en términos de la cantidad de conexiones simultáneas.

Bajo el esquema basado en eventos, la discusión en términos de recursos es diferente. Es cierto que se disminuye la huella de memoria bajo carga y se desperdicia menos tiempo de CPU para cambiar de contexto. Pero los desafíos que se posan sobre el hilo único de una arquitectura basada en eventos son contrarios a aquellos en una arquitectura basada en hilos. En una arquitectura de eventos, el CPU se convierte en el principal limitante aparente de una aplicación de red. Es posible que la cola de eventos comience a sobrecargarse y el hilo de procesamiento de eventos no pueda responder a la par.

Efectos sobre Rendimiento

Se han escrito reportes detallados de por qué la evaluación comparativa de arquitecturas de servidores es extremadamente difícil. En primer lugar, es difícil establecer puntos de comparación entre una arquitectura y otra, debido a que su funcionamiento y gestión de recursos es distinto.

Casi siempre se llega a la conclusión que los modelos arquitectónicos se pueden usar para construir servidores escalables, siempre y cuando se realice un ajuste y una configuración adecuada.

Las métricas que se utilizan en benchmarks de rendimiento de comparación de arquitecturas incluyen cantidades como solicitudes por segundo, tiempo de respuesta, ancho de banda de entrada y salida, entre otras. En general, las métricas obtenidas en las distintas arquitecturas bajo condiciones equivalentes son muy parecidas.

Es común que una aplicación se beneficie de un modelo híbrido (SEDA) para mejorar el rendimiento.

Conclusión

La elección de una arquitectura de servidor como parte de las decisiones de arquitectura, es crítica por el impacto a largo plazo sobre la escalabilidad, mantenibilidad y fiabilidad de un sistema. Todas las arquitecturas son confiables y se pueden construir servicios de alto rendimiento a partir de cada una de ellas, ajustado al grado de la configuración adecuada. Es de vital importancia conocer el funcionamiento, así como las virtudes y desventajas de cada arquitectura para adquirir conocimiento suficiente sobre servidores.

Bibliografía

1. Benjamin Erb. (2012). Server Architectures. 2021, de Ulm University Sitio web: https://berb.github.io/diploma-thesis/original/042_serverarch.html
2. Vaardan Driola, Cedric Chin Shen Wang. (2015). Threads vs Events for Server Architectures. 2021, de National University of Singapore Sitio web: http://vellvisher.github.io/papers_reports/doc/Threads_vs_Events_Server_Architectures.pdf
3. Gurudas Somadder, Dorina Petriu. (1997). Performance Measurements of Multi-Threaded Servers in a Distributed Environment . 2021, de Carleton University Sitio web: https://link.springer.com/content/pdf/10.1007/978-0-387-35188-9_12.pdf
4. David Pariag, Tim Brecht, et al. (2007). Comparing the Performance of Web Server Architectures. 2021, de University of Waterloo Sitio web: <http://course.ece.cmu.edu/~ece845/docs/pariag-2007.pdf>
5. Nimesh Chhetr. (2016). A Comparative Analysis of Node.js (Server-Side JavaScript). 2021, de St. Cloud State University Sitio web: <https://core.ac.uk/download/pdf/232792216.pdf>
6. D. Brent Chapman & Elizabeth D. Zwicky. (1997). Packet Filtering. 2021, de O'Reilly & Associates Sitio web: http://web.deu.edu.tr/doc/oreily/networking/firewall/ch06_01.htm
7. Gordon McMillan. (2021). Socket Programming HOWTO. 2021, de Python Sitio web: <https://docs.python.org/3/howto/sockets.html>
8. Douglas C. Schmidt. (1999). Reactor. 2021, de Siemens AG Sitio web: <http://www.laputan.org/pub/sag/reactor.pdf>