CIMAT

Arquitectura Cliente-Servidor

Reporte

Felipe Miramontes Romero

9/13/2014

El siguiente documento es un reporte acerca de la arquitectura Cliente-Servidor, se pretende explicar cuál es funcionamiento de mismo así como sus ventajas y desventajas, ejemplos simples para una mejor comprensión y un ejercicio práctico desarrollado en PHP.

TABLA DE CONTENIDO

ARQUITECTURA CLIENTE-SERVIDOR

INTRODUCCIÓN

En el siguiente reporte se pretende explicar a fondo el funcionamiento y la lógica utilizada por la arquitectura cliente-servidor, cabe mencionar que es un modelo utilizado para el desarrollo de aplicaciones distribuidas en el cual las tareas son repartidas entre cada uno de los proveedores de servicios o recursos, la capacidad proceso es compartida entre los clientes y servidores, en este tipo de red los nodos clientes se encuentran conectados al nodo servidor en el cual se centralizan los recursos y las aplicaciones que son puestas a disposición de los clientes cada vez que son solicitadas.

EVOLUCIÓN DE LA ARQUITECTURA CLIENTE-SERVIDOR

Para comprender como ha evolucionado la arquitectura cliente-servidor a través de los años es necesario seccionar la historia de acuerdo a las técnicas empleadas para la administración de datos., es así como y de acuerdo a (***Calvo***, *Jorge Mario*) son identificadas las siguientes eras:

- La era de la computadora central; basado en el uso de terminales remotas, una computadora central se encargaba de prestar servicios a grupos exclusivos de usuarios.
- La era de las computadoras dedicadas; en esta época cada uno de los servicios empleaba su propia computadora, permitiendo a los usuarios conectarse directamente a dicho servicio.
- La era de la conexión libre; ocurre cuando las computadoras de escritorio aparecen de forma masiva, esto permitió que los procesos de cálculo se realizaran en propio escritorio del usuario. Los recursos son transmitidos por medio de recursos magnéticos o por transcripción.
- La era del cómputo a través de redes; aparición de redes de computadoras, todos los usuarios pueden acceder a la información contenida en todas las computadoras conectadas en la red.

- La era de la arquitectura cliente servidor; en esta arquitectura los usuarios producen una demanda de información a cualquier computadora llamada servidor la cual tiene como fin primordial proporcionar datos a quien los requiera, en este modelo el usuario tiene la libertad de generar peticiones y procesar información según le sea conveniente.

CARACTERIZACIÓN DE LA ARQUITECTURA C/S

Se puede definir como cliente a un dispositivo remitente de una solicitud, el cual tiene las siguientes características: inicia las solicitudes y tiene un papel activo en la comunicaciones se le conoce como dispositivo amo o maestro, espera por las respuestas del servidor, puede enviar peticiones a varios servidores a la vez, interactúa con usuarios por medio de una interfaz gráfica.

Es posible definir como servidor al receptor de solicitudes envías por clientes el cual posee las siguientes características: espera por las solicitudes de clientes y son conocidos como dispositivos esclavos, reciben, procesan y envían la respuesta, aceptan conexiones por parte de un gran número de clientes, no es frecuente la interacción con usuarios finales.

FUNCIONAMIENTO

Para explicar de manera lógica el funcionamiento de la arquitectura se realizara la mención de los pasos necesarios para lograr una iteración entre un cliente y un servidor.

1. El servidor es ejecutado o puesto en marcha.*
2. El servidor se encuentra en espera de las peticiones de los dispositivos clientes.
3. El cliente es ejecutado o puesto en marcha.*
4. El cliente envía la petición.
5. La petición es recibida por el servidor.
6. El servidor procesa la petición del cliente.
7. El servidor envía la información al cliente.

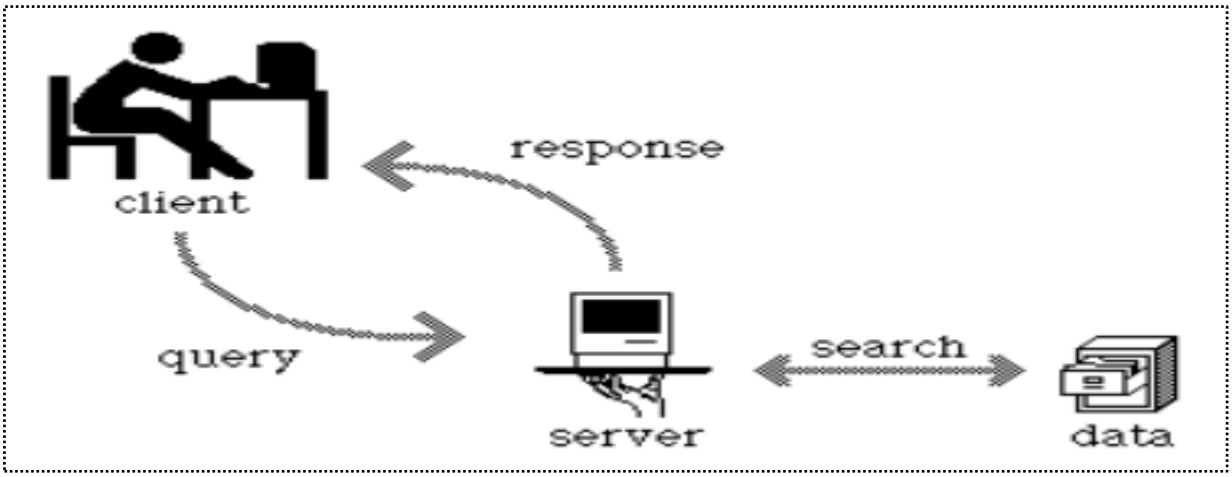

Imagen 1, http://www.cse.iitk.ac.in/users/dheeraj/cs425/lec17.html,

COMPARACIÓN CON ARQUITECTURA CLINTE-COLA-CLIENTE

A diferencia de la arquitectura C/S (Cliente-Servidor) la arquitectura C/C/C (Cliente-Cola-Cliente) permite a todos dispositivos actuar como clientes simples, mientras tanto el servidor actúa como almacenamiento de peticiones a manera de cola, esta arquitectura ha dado paso a P2P (Peer To Peer).

VENTAJAS DE LA ARQUITECTURA CLIENTE-SERVIDOR

1. El control es centralizado, tanto los accesos como los recursos y la integridad de la información se encuentran bajo control del servidor de tal manera que un programa cliente con defectos o alguno no autorizado no pueda dañar el sistema, de esta manera también es sencillo actualizar los datos y recursos mejor que redes P2P.
2. A la hora de pensar en escalabilidad se puede aumentar la capacidad de operación de cada una de las partes por separado, cualquier parte o elemento puede ser mejorado en cualquier momento, es posible añadir nuevas partes a la red ya sean clientes o servidores.
3. El mantenimiento es económico, debido a la estructura distribuida de las responsabilidades de los servidores es factible realizar modificaciones mientras los clientes no se ven afectados a esta independencia se le conoce como encapsulación.
4. Algunas tecnologías para el paradigma C/S aseguran, seguridad, amigabilidad y fácil empleo.

DESVENTAJAS DE LA ARQUITECTURA CLIENTE-SERVIDOR

1. El tráfico de peticiones es uno de los principales problemas con esta arquitectura, cuando una cantidad alta de clientes envían simultáneamente requisiciones de información a un mismo cliente puede causar problemas a este, al contrario de las redes P2P ya que cada dispositivo en la red juega el papel de servidor.
2. Carece de robustez, cuando un servidor falla las peticiones de los clientes no serán de ninguna manera atendidas, a diferencia de las redes P2P pues en estas cuando un servidor falla los demás atienden la petición.
3. Los recursos de hardware y software son determinantes ya que pueden estar limitados para atender cierta cantidad de clientes, mientras más clientes es necesario atender el coste de infraestructura es mayor.
4. Los dispositivos clientes no disponen en ningún momento de los recursos contenidos por el servidor, no es posible modificar el disco duro de los clientes cuando hablamos de una aplicación web.

EJEMPLOS DE ARQUITECTURA CLIENTE-SERVIDOR

El acceso en línea a los servicios de banca por medio de un web browser (El cliente), el cliente inicia una petición al servidor del banco, el cliente inicia sección con credenciales válidas que se encuentran almacenadas in una base de datos, el servidor web accede a la base de datos como cliente, la aplicación del servidor interpreta los datos y los regresa al cliente aplicando la lógica de negocio del propio banco.

El acceso a información privilegiada por medio de sockets a través de un protocolo, una IP y un puerto específico, el cliente se conecta con el servidor para hacer requisiciones de información específica, utilizada para accesos rápidos y recurrentes a información activa donde se garantiza que cada uno de los octetos de información lleguen a su destino en el orden exacto en el cual fue transmitido además se evita que los datos adquieran errores y omisiones.

DESCRIPCIÓN DE CAPASIDADES PHP

PHP (Hippertext Preprocessor) es un multiparadigma, multiplataforma, imperativo, que permite programación orientada a objetos, procedural y reflexiva, diseñado por Rasmus, Lerdford en 1995.

Definido como- un lenguaje de programación de uso general de código del lado servidor- orientado al desarrollo web de contenido dinámico, considerado flexible, potente y de alto rendimiento por varios autores, lenguaje parte del software libre bajo su propia licencia PHP la cual es incompatible con GNU. Como dato informativo: incluye manejo de excepciones desde PHP 5.

PHP, permite usar aplicaciones como Zend Framework y posee extensiones como: sockets, Match, Stat, entre otras.

Debido a las características mencionadas es por lo que se le ha seleccionado como lenguaje para el desarrollo de los siguientes programas.

CARATERÍSTICAS DE LOS PROGRAMAS A DESARROLLAR

El programa server.php fue realizado para emular el papel del servidor en la arquitectura cliente-servidor: ser ejecutado, esperar por la conexión con un ente “Cliente”, esperar, recibir, procesar y regresar la información al cliente, a continuación se muestra el código.

```php
<?php

//Tiempo de conexion del socket, (0)= El socket no se cierra por efectos de termino de tiempo
set_time_limit(0);

// creamos el array que contiene la informacion que queremos buscar
$contenido = array('Lupita' => 4921210332, 'Grecia'=> 4921210331, 'Felipe'   => 4921210333,
                   'Roberto'=> 4921210330, 'Oyuki' => 4921210329, 'Juan'     => 4921210328,
                   'Brisia' => 4921210318, 'Mayra' => 4921210326, 'Sustaita' => 4921210329,
                   'Tovar'  => 4921210324, 'Segio' => 4921210322, 'Mars'     => 4921210321,
                   'Juanis' => 4921210320, 'Tavo'  => 4921210319, 'Luis'     => 4921210318);

//IP Adresss
$ip = '127.0.0.1';
//Puerto
$puerto = '10001';

/* Creación del socket
AF_INET Especifca el protocolo de la conexion (AF_INET - AF_INET6 - AF_UNIX)
SOCK_STREAM indica que los bites de la conexion seran eviados por medio de _STREAM
*/
$socket = socket_create(AF_INET, SOCK_STREAM, getprotobyname('tcp'));

//Vinculacion de puerto con IP
socket_bind($socket, $ip, $puerto) or die ('No se puede vincular el puerto a la IP');

```

```php
//Mensaje para identificar posible error
echo socket_strerror(socket_last_error());

//Iniciamos la espera de peticiones
socket_listen($socket);
 $i=0;

   while(1){
    //Aceptamos conexiones entrantes
    $cliente[++$i] = socket_accept($socket);

    //Leemos la peticion enviada
    $input = socket_read($cliente[$i], 1024);

    //Ejecutamos la peticion quitando espacios en blando y agregando la informacion requerida
    $peticion = preg_replace("[ \t\r\n]", "", $input);
    $inf = $contenido[$peticion];
    echo "\t\r\n";
    echo "Peticion recibida de parte del cliente: $peticion";
    echo "\t\r\n";
    echo "Respuesta enviada al cliente: $inf";
    echo "\t\r\n";

```

```php
    if(array_key_exists($peticion, $contenido)){
        //Se busca la informacion requerida en nuestro almacen de datos tipo Diccinario de datos
        $info = $contenido[$peticion];
    }else{
        //Mensaje de advertencia sobre información inexsistente
        $info = "<Error: ¡Error el usuario no existe!>";
    }
    //Escribimos el rultado que sera retribuido al cliente
    socket_write($cliente[$i], $info . "\n\r", 1024);
    //Se cierra la conexion
    socket_close($cliente[$i]);
}
//Se cietta el proceso goblan de tipo socket
socket_close($socket);
?>
```

El programa client.php fue realizado para emular el papel del cliente en la arquitectura cliente-servidor: ser ejecutado, enviar peticiones y esperar por la respuesta.

```php
<?php
while(1){

    //Se despliega mensaje de bienvenida
    echo "\t\r\n";
    echo "<Estado de Conexion: Establecida>";
    echo "\t\r\n";
    echo "<Estado del Servidor: En espera de una peticion>";
    echo "\t\r\n";
    echo "Ingrese un nombre de usuario o escriba el comando 'Quit' para salir: ";

    //Obtenemos la peticion del usuario
    $user = trim(fgets(STDIN));

    //Se valida el conmando "Quit" de salida
    if ($user == 'Quit')
    {
        exit;
    }

    // Mensaje que despliega el nombre de usuario consultado en caso de que exista.
    echo "\t\r\n";
    echo "Peticion enviada, $user ";
    echo "\t\r\n";

    //Creacion de socket con caracteristicas TCP
    $socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

```php

        //Se verifica que el socket se creo con exito.
        if($socket === FALSE)
        {
            echo '<ERROR: error en la conexion de sockets!>';
        }
        else
        {
            //Conexion con el socket servidor (OBJETO, IP_ADRESS, PUERTO)
            $resultado = socket_connect($socket, '127.0.0.1', '10001');

            if($resultado === FALSE)
            {
                echo '<ERROR: error en la conexion de sockets!>';
            }
            else
            {
               //Se envia la peticion de información
               socket_write($socket, $user, strlen($user));
               // Se recibe la información
               $info = socket_read($socket, 1024);
               // Se imprime la información recibida
               echo "Respuesta recibida, $info";
            }
        }
    }
?>
```

Los programas anteriormente descriptos funcionas por medio de una entrada del programa client.php, el cual envía un nombre de usuario registrado, el programa server.php recibe la petición y envía una respuesta con el número de teléfono del usuario ingresado por el server. El código puede ser encontrado en el siguiente repositorio: https://github.com/felipemiramontesr/client_server_sockets_php

EJEMPLOS EXTRA DE SOCKETS PHP

```
Object oriented example of the server

#!/usr/bin/env php
<?php

class MySocketServer
{
    protected $socket;
    protected $clients = [];
    protected $changed;

    function __construct($host = 'localhost', $port = 9000)
    {
        set_time_limit(0);
        $socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
        socket_set_option($socket, SOL_SOCKET, SO_REUSEADDR, 1);

        //bind socket to specified host
```

```
        socket_bind($socket, 0, $port);
        //listen to port
        socket_listen($socket);
        $this->socket = $socket;
    }

    function __destruct()
    {
        foreach($this->clients as $client) {
            socket_close($client);
        }
        socket_close($this->socket);
    }

    function run()
    {
        while(true) {
            $this->waitForChange();
            $this->checkNewClients();
            $this->checkMessageRecieved();
            $this->checkDisconnect();
        }
    }

    function checkDisconnect()
    {
        foreach ($this->changed as $changed_socket) {
            $buf = @socket_read($changed_socket, 1024, PHP_NORMAL_READ);
            if ($buf !== false) { // check disconnected client
                continue;
            }
            // remove client for $clients array
            $found_socket = array_search($changed_socket, $this-
>clients);
            socket_getpeername($changed_socket, $ip);
            unset($this->clients[$found_socket]);
            $response = 'client ' . $ip . ' has disconnected';
            $this->sendMessage($response);
        }
    }

    function checkMessageRecieved()
    {
        foreach ($this->changed as $key => $socket) {
            $buffer = null;
            while(socket_recv($socket, $buffer, 1024, 0) >= 1) {
                $this->sendMessage(trim($buffer) . PHP_EOL);
                unset($this->changed[$key]);
                break;
            }
        }
    }

    function waitForChange()
```

```
    {
        //reset changed
        $this->changed = array_merge([$this->socket], $this->clients);
        //variable call time pass by reference req of socket_select
        $null = null;
        //this next part is blocking so that we dont run away with cpu
        socket_select($this->changed, $null, $null, null);
    }

    function checkNewClients()
    {
        if (!in_array($this->socket, $this->changed)) {
            return; //no new clients
        }
        $socket_new = socket_accept($this->socket); //accept new socket
        $first_line = socket_read($socket_new, 1024);
        $this->sendMessage('a new client has connected' . PHP_EOL);
        $this->sendMessage('the new client says ' . trim($first_line) .
PHP_EOL);
        $this->clients[] = $socket_new;
        unset($this->changed[0]);
    }


    function sendMessage($msg)
    {
        foreach($this->clients as $client)
        {
            @socket_write($client,$msg,strlen($msg));
        }
        return true;
    }
}

(new MySocketServer())->run();
?>
```

Este ejemplo muestra un simple, único cliente HTTP. Simplemente se conecta a una página, envía una petición HEAD, repite la réplica, y sale.

```
<?php
error_reporting(E_ALL);

echo "<h2>TCP/IP Connection</h2>\n";

/* Obtener el puerto para el servicio WWW. */
$service_port = getservbyname('www', 'tcp');

/* Obtener la dirección IP para el host objetivo. */
$address = gethostbyname('www.example.com');

/* Crear un socket TCP/IP. */
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
if ($socket === false) {
```

```
    echo "socket_create() falló: razón: " . socket_strerror(socket_last_e
rror()) . "\n";
} else {
    echo "OK.\n";
}

echo "Intentando conectar a '$address' en el puerto '$service_port'...";
$result = socket_connect($socket, $address, $service_port);
if ($result === false) {
    echo "socket_connect() falló.\nRazón: ($result) " . socket_strerror(s
ocket_last_error($socket)) . "\n";
} else {
    echo "OK.\n";
}

$in = "HEAD / HTTP/1.1\r\n";
$in .= "Host: www.example.com\r\n";
$in .= "Connection: Close\r\n\r\n";
$out = '';

echo "Enviando petición HTTP HEAD ...";
socket_write($socket, $in, strlen($in));
echo "OK.\n";

echo "Leyendo respuesta:\n\n";
while ($out = socket_read($socket, 2048)) {
    echo $out;
}

echo "Cerrando socket...";
socket_close($socket);
echo "OK.\n\n";
?>
```

CONCLUSIONES

La arquitectura cliente-servidor es un modelo bastante simple, las ventajas que este ofrece pueden ser aprovechadas al máximo cuando se tiene un conocimiento pleno sobre el diseño de nuestra aplicación distribuida, debido a su economía es posible implementar soluciones escalables con bajo presupuesto, las capacidades que ofrece sobre la integridad de los datos es sin duda alguna bastante atractiva cuando se quiere realizar aplicaciones seguras de bajo perfil, por otro lado, carece de robustez pues cuando el nodo server falla las conexiones de trabajo dejan de funcionar, siempre es necesario considerar la máxima cantidad de clientes que década servidor podrá atender ya que esto está directamente relacionado con el costo de infraestructura.

En mi propia experiencia puedo decir que la arquitectura cliente servidor es bastante confiable tomando en cuenta sus ventajas y desventajas, sin embargo hoy en día es más fiable utilizar una arquitectura como P2P (Peer To Peer).

FUENTES

Nieh, Jason; Novik, Naomi; Yang, S. Jae (December 2005). *A Comparison of Thin-Client Computing Architectures* (PDF). New York: Network Computing Laboratory, Columbia University . Retrieved 30 November 2013, from https://www.nomachine.com/documentation/pdf/cucs-022-00.pdf

Yon sheng, H.; Xiaoyu, T.; Zhongbin, T. (2013). An Optimization Model for the Interconnection among Peers of the P2P Network. *Journal of Applied Sciences* **13** (5): 700. doi:10.3923/jas.2013.700.707.

Barros, A. P.; Dumas, M. (2006). The Rise of Web Service Ecosystems. *IT Professional* **8** (5): 31. doi:10.1109/MITP.2006.123.

National Renewable Energy Laboratory. (2008). *Biofuels*. Retrieved May 6, 2008, from http://www.nrel.gov/learning/re_biofuels.html

Ecured, conocimiento con todos y para todos, (2014). Arquitectura cliente-servidor. Retrieved Sep 13, 2014, from http://www.ecured.cu/index.php/Arquitectura_Cliente_Servidor

Desarrollo Web, Manual de iniciación al a programación, (2014). Arquitectura cliente-servidor. Retrieved Sep 13, 2014, from http://www.desarrolloweb.com/articulos/arquitectura-cliente-servidor.html

Flores, F. (2014, september 13). Fausto videos blogspot, historia cliente-servidor. Message posted to http://faustovideos.blogspot.mx/p/historia-cliente-servidor.html

Php,(2014). *Ejemplos sockets.* Retrieved Sep 13, 2014, from http://php.net/manual/es/sockets.examples.php