

SOCKET E PHP

Introduzione.

I socket sono i meccanismi principali che permettono la trasmissione e la ricezione di dati in una rete. Rappresentano la porta tra il processo dell'applicazione e il protocollo di trasmissione (TCP,UDP). I socket permettono allo sviluppatore di introdurre un sistema di comunicazione, attraverso la rete, all'interno della propria applicazione, grazie all'uso di API (Application Programming Interface). In questo modo lo sviluppatore si preoccupa principalmente di COSA l'applicazione deve trasmettere, rispetto a COME deve trasmettere.

Il PHP mette a disposizione delle funzioni molti simili a quelle utilizzate nel linguaggio C (non per niente è la sua controparte interpretata) che danno la possibilità di creare, distruggere e gestire un socket.

Le funzioni principali sono:

- **socket_create** — Crea un socket (punto terminale di una comunicazione).
- **socket_close** — Chiude una risorsa di tipo socket
- **socket_accept** — Accetta una connessione su un socket
- **socket_bind** — Bind di un nome ad un socket
- **socket_connect** — Inizia una connessione su un socket
- **socket_getpeername** — Interroga il lato remoto di un dato socket per ottenere o la combinazione host/porta od un percorso Unix, in base al tipo di socket
- **socket_getsockname** — Interroga il lato locale di un dato socket e restituisce o la combinazione host/porta oppure un percorso Unix in base al tipo di socket
- **socket_last_error** — Restituisce l'ultimo errore su un socket.
- **socket_listen** — Attende una richiesta di connessione su un socket
- **socket_read** — Legge fino ad un massimo di byte predefiniti da un socket
- **socket_recv** — Riceve i dati da un socket collegato
- **socket_send** — Invia i dati ad un socket collegato
- **socket_set_option** — Valorizza le opzioni per un socket
- **socket_strerror** — Restituisce una stringa con la descrizione dell'errore.
- **socket_write** — Scrive su un socket.

Le funzioni dovrebbero girare correttamente sia su sistemi Unix Like che su sistemi Windows.

Gestione Errori

Come la maggior parte delle funzioni, le precedenti, in caso di errore restituiscono un *E_WARNING* che descrive l'errore. Spesso però come si può leggere su Php.net ci possono essere casi in cui ci viene restituito un errore inaspettato. Per esempio la funzione `socket_read()` potrebbe improvvisamente generare un errore e restituire l'*E_WARNING* successivamente ad una disconnessione improvvisa del socket. Solitamente si sopprime l'errore restituito con il simbolo `@` di fronte alla funzione e si cerca di intercettarlo grazie alla funzione `socket_last_error()`. Si può anche usare il codice di errore intercettato con la funzione `socket_strerror()` in modo da farsi restituire una stringa descrittiva dell'errore.

Comunicazione attraverso i socket

Se due applicazioni devono comunicare tra loro, come abbiamo detto precedentemente, utilizzano i socket. Le applicazioni, o meglio la parte che si occupa della comunicazione, può utilizzare diversi modi per interfacciarsi con le altre applicazioni in modo da scambiare dati. L'architettura più diffusa è il Client-Server, ovvero una applicazione è sempre in attesa di essere contattata (server) e un'altra è quella che contatta (client). Il server sarà quindi sempre in ascolto su una determinata porta, e nel momento in cui gli arriva una richiesta di connessione, questo potrà accettarla ed iniziare lo scambio dati, o rifiutarla in base alle policy create dallo sviluppatore. Iniziamo quindi a vedere come utilizzare i socket con il PHP in modo da creare un client e un server per l'invio e la ricezione di file. Il client leggerà il file e lo invierà al server che si occuperà di scriverlo su disco.

Client

Analizziamo passo passo le azioni che il client deve compiere. Innanzitutto prende il nome del file da inviare, l'indirizzo del server e la porta su cui è in ascolto. Dopodiché dovrà verificare che il file esista e aprirlo in lettura. Se non si sono riscontrati errori durante queste azioni si può inizializzare il socket, e connetterlo al server. Infine leggere un tot di byte dal file, metterlo in un buffer e inviarlo al server, questo procedimento deve essere effettuato fino a quando il puntatore nel file arriva alla fine del file stesso. Fatto questo si avvisa il server che l'invio è completo e che si può chiudere la connessione.

Iniziamo dunque...

Verifichiamo l'esistenza del file e apriamolo.

```
<?php
set_time_limit(0); // Evita che dopo un tot di tempo l'interprete php blocchi lo script come da php.ini
if( !( $file = @fopen( "file", "r" ) ) ) {
    echo "File open error\n.";
    exit;
}
```

Se il file non esiste, o non abbiamo i permessi per aprirlo, la funzione fopen restituirà FALSE, quindi usciremo dal programma con un errore.

Il client si preoccupa di contattare il server, quindi necessita dell'IP del server e della porta su cui è in ascolto. Inoltre creerà un socket che successivamente collegherà al server:

```
$server = array( 'address' => '192.168.0.102', 'port' => 7001 );
$sock = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
...
```

Come possiamo vedere la funzione socket_create che si occupa di creare opportunamente un socket necessita di alcuni parametri. Ovvero bisogna decidere il tipo di protocollo da utilizzare: TCP o UDP. Non mi soffermerò molto su questi protocolli, dando per scontato che voi li conosciate. La funzione necessita per la precisione di 3 parametri: famiglia del protocollo, tipo di comunicazione, specifico protocollo per quel dominio.

Le famiglie di protocolli sono AF_INET che racchiude tutti i protocolli internet basati su IPV4 (TCP, UDP,..), AF_INET6 che sono basati su IPV6, ed infine AF_UNIX che rappresenta dei protocolli usati per le comunicazioni locali tra processi (IPC). Il tipo di comunicazione specifica come deve avvenire la comunicazione, ovvero se si debbano usare i datagram o se si vuole una connessione di qualità, o ancora se si vogliono utilizzare i socket in versione raw (utile per costruirsi un protocollo). Infine c'è il tipo di protocollo, noi abbiamo usato il TCP, ma per AF_INET è possibile anche UDP.

Vediamo ora di connetterci al server:

```
if( !( $connessione = @socket_connect( $sock, $server['address'], $server['port'] ) ) ) {
    echo socket_strerror(socket_last_error($sock));
    exit;
}
```

Come potete notare ho inserito anche la gestione degli errori. Non molto avanzata, ma c'è.

Iniziamo ora a leggere il file 1024 byte per volta, li inseriamo in un buffer e li inviamo pari pari al server tramite la connessione appena creata. Ovviamente dovremo controllare anche se la lettura del file termina, in modo da avvertire il server di chiudere il file e la connessione.

```

echo "Uploading file.";

do {
    if( feof( $file ) ) {
        echo "send complete\n";
        socket_write( $sock, "--@END@--" );
        break;
    }
    if( !( $pkt = @fread( $file, 1024 ) ) ) {
        echo socket_strerror(socket_last_error($sock));
        exit;
    }

    if( !( @socket_write( $sock, $pkt, 1024 ) ) ) {
        echo socket_strerror(socket_last_error($sock));
        exit;
    }

    echo ".";

    if( !( $rcv = @socket_read( $sock, 1024 ) ) ) {
        echo socket_strerror(socket_last_error($sock));
        exit;
    }
    //echo $rcv."\n";
} while( true );

echo "DONE\n";

fclose( $file );
socket_close( $sock );
echo "Exiting...\n";

?>

```

Vediamo ora il server.

SERVER

Come per il client vediamo le operazione che il server deve compiere affinché possa aprire un socket e ricevere un file. Inizialmente creerà il socket come il client, ma a differenza di quest'ultimo non cercherà di connettersi, bensì dovrà attendere le connessioni, ovvero dovrà mettersi in listening su una determinata porta (7001). Inoltre aprirà un file in scrittura. Quando riceverà la richiesta di connessione da un client, la accetterà o meno, prenderà i dati in arrivo sul socket, li inserirà in un buffer e li scriverà nel file aperto in precedenza. Questo fino a quando non vedrà arrivare la stringa di fine invio (--@END@--). A questo punto chiuderà socket e file ed uscirà. Passiamo quindi al codice.

```

<?php
set_time_limit(0);
if( !( $file = @fopen( "binario", "w" ) ) ) {
    echo "File open error.\n";
    exit;
}
if( !( $sock = @socket_create( AF_INET, SOCK_STREAM, SOL_TCP ) ) ) {
    echo socket_strerror(socket_last_error($sock));
    exit;
}

```

Facciamo il bind dell'indirizzo sul socket, ovvero mettiamo in ascolto il socket sull'interfaccia di rete che si affaccia sulla rete con l'ip 192.168.0.101 e sulla porta 7001 (ricordo che su sistemi Unix Like le porte al di sotto della 1024 necessitano di un'applicazione lanciata con privilegi root per poterle utilizzare). Successivamente diciamo al socket di mettersi in ascolto con `socket_listen()`.

```
if( !( @socket_set_option($sock, SOL_SOCKET, SO_REUSEADDR, 1 ) ) ) {
    echo socket_strerror(socket_last_error($sock));
    exit;
}
if( !( @socket_bind( $sock, '192.168.0.101', 7001 ) ) ) { // Associamo un ip della macchina e la porta al socket
    echo socket_strerror(socket_last_error($sock));
    exit;
}
if( !( @socket_listen( $sock ) ) ) { // mettiamo il socket in ascolto
    echo socket_strerror(socket_last_error($sock));
    exit;
}
```

La funzione `socket_set_option()` viene utilizzata per configurare alcune opzioni del socket, come per esempio il timeout in lettura o scrittura, il riutilizzo dell'indirizzo (il SO memorizza la porta e l'indirizzo su cui si effettua il bind e lo riserva per un determinato periodo di tempo, quindi capita alle volte che il bind fallisca anche se la nostra applicazione è stata chiusa pochi istanti prima), ecc...

Ora il nostro server dovrà attendere che un client si connetta, accetterà la richiesta di connessione e inizierà a leggere dal socket fino a quando non incontrerà la stringa di fine invio. Siccome in invio il client scrive sul socket pacchetti di 1024 byte alla volta, leggeremo pacchetti di 1024 byte in ricezione.

N.B.: La funzione `socket_getpeername()` serve a prendere l'ip del client, per mostrarlo a video con `echo`.

```
while( true ) {
    if( !( $client = @socket_accept( $sock ) ) ) { // funzione bloccante: finchè non arriva una richiesta si
    rimane bloccati qui
        echo socket_strerror(socket_last_error($sock));
        exit;
    }
    if( !( @socket_getpeername( $client, $addr ) ) ) { // prendiamo l'IP del client che si è connesso
        echo socket_strerror(socket_last_error($sock));
        exit;
    }
    echo "Client connection from: $addr\n";
    echo "Receiving file .";
    if( !( $rcv = @socket_read( $client, 1024, PHP_BINARY_READ ) ) ) {
        echo socket_strerror(socket_last_error($sock));
        exit;
    }
    do {
        if( !( @fwrite( $file, $rcv, 1024 ) ) ) {
            echo socket_strerror(socket_last_error($sock));
            exit;
        }
        if( !( @socket_write( $client, "ok" ) ) ) {
            echo socket_strerror(socket_last_error($sock));
            exit;
        }
    }
}
```

```
        if( !( $rcv = @socket_read( $client, 1024, PHP_BINARY_READ ) ) ) {
            echo socket_strerror(socket_last_error($sock));
            exit;
        }
        echo ".";
    } while( $rcv != "--@END@--" );
    echo "DONE\n";
    socket_close( $client );
    break;
}
fclose( $file );
socket_close( $sock ); // prepariamoci per la nanna...
echo "Exiting...\n";
?>
```

CONCLUSIONE

Ok, ora tutto dovrebbe funzionare, per provare gli script lanciate il server con `php server.php` e successivamente il client. Ovviamente sostituite gli IP con quelli che utilizzate e cambiate il nome del file da inviare.

I sorgenti completi di server e client si possono trovare sul mio sito <http://www.tuxmealux.net> Precisamente verranno resi disponibili nel forum.

Spero che questo semplice how to vi possa essere utile per introdurvi nel mondo del php. Per aiuti, dubbi, insulti e altro ci sentiamo su <http://www.tuxmealux.net/forum> o su <http://www.rbt-4.net>

Fonti utilizzate:

<http://www.php.net>