

Application socket client/serveur en PHP

Partie II – Utilisation des sockets

I – Définition d'une socket

Une socket est un point de communication entre deux ou plusieurs entités. Une connexion typique réseau est composée de deux sockets : une qui joue le rôle de client et l'autre celui du serveur.

Source : <http://fr.php.net/manual/fr/book.sockets.php>

II – Création d'un serveur

II.a – Introduction

II.b – Création du `function.inc.php`

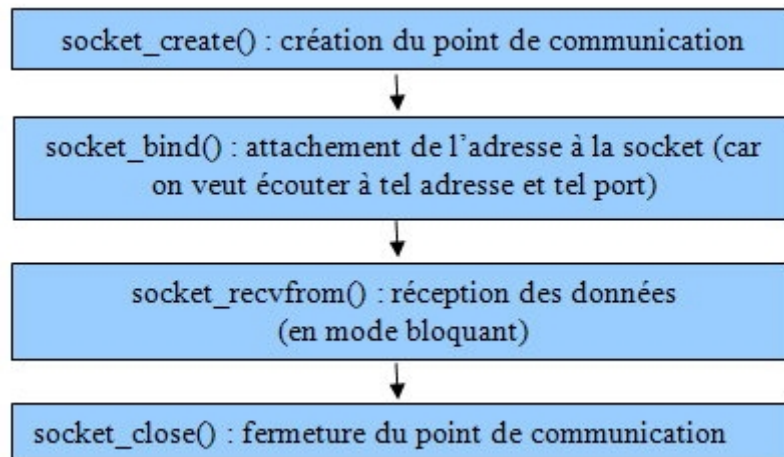
II.c – Création du fichier `serveur.php`

II.a – Introduction

Le principe est sensiblement le même que celui utilisé avec les stream sockets à une différence près : l'attachement de l'adresse à la socket.

Dans ces sockets "traditionnelles" il faut spécifier à quel adresse la socket devra écouter.

Nous utiliserons donc le principe suivant pour créer notre serveur :



On utilisera toujours un code structuré comme ceci :

```

/function.inc.php
-- serveur.php
-- client.php
  
```

Ceci va nous permettre de séparer notre code et de ne pas avoir de fonctions redondantes (comme celle de création de socket ou de fermeture).

II.b – Création du function.inc.php

Code commun dans function.inc.php au serveur et au client :

```

function creation_socket()
{
    $socket = socket_create(AF_INET , SOCK_DGRAM , SOL_UDP);
    /*
        AF_INET est la famille d'adresses Internet,
        SOCK_DGRAM correspond au type de sockets, ici socket mode
        non connecté ( UDP )
        SOL_UDP est le protocole utilisé, UDP donc.
    */

    if($socket < 0)
    {
        echo 'Erreur de création de la socket.';
        exit(1);
    }

    /* Retourne le descripteur de la socket */
    return $socket;
}
  
```

Cette socket n'a besoin que du protocole de transport, sa famille et le type de socket.

Si \$socket est négatif, c'est que la socket n'a pas été créée, on retourne donc un message et on ferme le programme.

L'erreur générée peut être lu grâce à la fonction [socket_last_error\(\)](#), cependant cette fonction ne retournera que le code d'erreur, il faudra utiliser [socket_strerror\(\)](#) sur cette fonction pour

avoir un résultat lisible et compréhensible.

Puisque nous sommes dans la partie commune, nous allons également créer la fonction de fermeture du point de communication :

```
function fermeture_socket($socket)
{
    /* Fermeture de la socket et détruit les ressources utilisées par
    celle-ci. */
    socket_close($socket);
}
```

Code servant au serveur :

Attachement de la socket réception des données.

Attachement de la socket :

```
function attachement_socket_addr($socket, $addr, $port)
{
    /* Sur un serveur il faut attacher l'adresse à une socket,
    ici on attachera à une adresse ip et un port */
    socket_bind($socket, $addr, $port);
}
```

La fonction servant à attacher une adresse à une socket est un bind, (idem pour des sockets C où la fonction est bind();)

Dans la fonction, on a besoin du descripteur de la taille, l'adresse et le port auxquelles on va écouté.

Réception des données :

```
define('TAILLE_MAX_DATA',1500);

function reception_donnees($socket, $port_ecoute)
{
    $from = "";
    $reception = socket_recvfrom($socket, $donnees, TAILLE_MAX_DATA, 0,
    $from, $port_ecoute);
    /* On précise à socket_recvfrom, le descripteur de la socket,
    la variable (buffer) permettant de recevoir les données,
    la taille des données à recevoir, l'adresse et le port de
    destination
    */

    /* socket_recvfrom() renvoi le nombre d'octets reçu ou -1 si non
    reçu */
    if($reception < 0)
        echo 'Erreur de réception des données';
    else
        echo 'Message reçu : '.$donnees;
}
```

Nous définissons une constante **TAILLE_MAX_DATA**, qui décrit la taille maximum des données que l'on recevra dans notre buffer, ici on la fixe à 1.5ko.

La fonction socket_recvfrom() a besoin du descripteur de la socket, le buffer de réception des données et sa taille, les flags (ici aucune option, donc 0).

Ici \$from correspond au paramètre name qui définit (en mode **non connecté**) l'adresse IP de l'hôte distant, on laisse une variable vide, ceci fonctionne sans l'adresse de l'hôte distant.

Et pour finir le port d'écoute.

Toujours le même principe d'erreur, si \$reception est négatif c'est que la réception a échoué, on affiche donc un message.

II.c – Création du fichier serveur.php

Une fois notre fonction.inc.php correctement créée, on peut créer notre serveur.php :

```
<?php
include('function.inc.php');

$addr = '127.0.0.1';
$port = '65000';
$socket = creation_socket();
attachement_socket_addr($socket, $addr, $port);

reception_donnees($socket, $port);

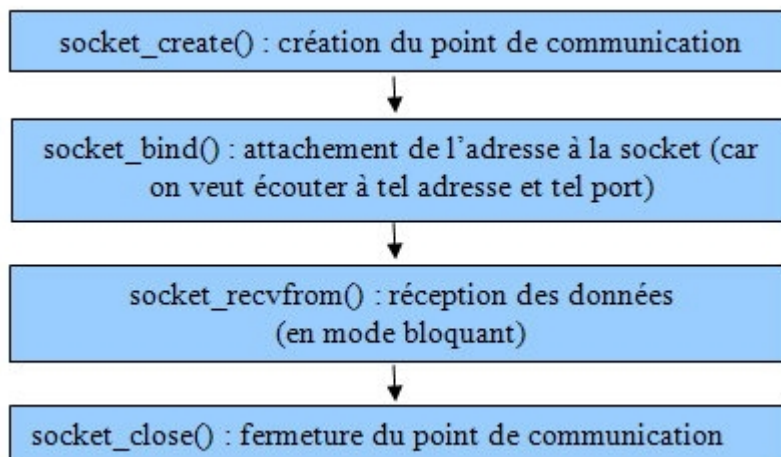
fermeture_socket($socket);
?>
```

Ce code est plus clair et plus simple à comprendre que si on avait un enchaînement de fonctions et de code permettant de récupérer les erreurs.

III – Création d'un client

Sur notre client, le principe est le même que pour le client en stream_socket, c'est-à-dire que l'on crée notre socket, on envoie nos données, on ferme le socket.

On peut représenter le principe avec ce schéma :



Puisque nos fonctions de création et de fermeture sont créées, il nous reste plus qu'à créer la fonction d'envoi des données dans function.inc.php :

```
function envoi_donnees($socket, $donnees, $addr_dest, $port_dest)
{
```

```

    $longueur_donnees = strlen($donnees); /* Calcul de la taille des
données */
    $envoi = socket_sendto($socket, $donnees, $longueur_donnees, 0,
$addr_dest, $port_dest);
    /* On précise à socket_sendto, le descripteur de la socket, les
données à envoyer,
sa taille, l'adresse et le port de destination
*/

    /* socket_sendto() renvoi le nombre d'octets transmis ou -1 si non
transmis */
    if($envoi < 0)
        echo 'Erreur d'envoi des données';
    else
        echo 'Données envoyées';
}

```

On a besoin du descripteur de la socket, des données à envoyer et de l'adresse/port de destination.

Même principe d'erreur, si l'envoi échoue la fonction renvoi -1 dans \$envoi, on affiche donc un message.

Une fois fonction.inc.php complété de cette fonction, on peut créer client.php :

```

<?php
include('function.inc.php');

$socket = creation_socket();

$addr_dest = '127.0.0.1';
$port_dest = '65000'; /* Port d'écoute du serveur */

$donnees = 'Bonjour je vous transmets un message !';
envoi_donnees($socket, $donnees, $addr_dest, $port_dest);

fermeture_socket($socket);
?>

```

IV - Activer les sockets

Sous EasyPHP, les sockets ne sont pas activés par défaut, il faut donc les activer :

Clique droit sur l'icône d'Easyphp > Configuration > PHP.

Ceci vous ouvre le fichier de configuration de PHP, chercher la ligne suivante :

```
;extension=php_sockets.dll
```

Puis enlevez le point virgule et enregistrez, puis redémarrez Apache.