TP1 : Processus, Signaux et Tubes

But du TP : Utilisation des processus, signaux et tubes de communication.

Pour ce TP nous travaillerons sur le serveur Web programmé l'année dernière.

Exercice 1 : le serveur Web a un fonctionnement itératif (les clients sont traités les uns après les autres). Ceci n'est pas très réaliste pour un vrai serveur Web. Proposez une version concurrente utilisant plusieurs processus avec l'appel système fork().

Exercice 2 : pendant l'exécution du serveur concurrent observez les processus du serveur Web avec la commande ps. Que constatez-vous ? Que peut-on faire pour remédier au problème ? En particulier, à quoi correspond le signal SIGCHLD ?

Exercice 3 (optionnel) : si l'on arrête (par Control-C ou avec un kill) le serveur Web, il y a un risque de perte de donnée et de blocage du numéro de port utilisé par l'application. Pour éviter cela, protégez le programme avec des gestionnaires de signaux qui le termineront correctement en cas de demande d'arrêt (SIGINT pour Control-C ou SIGTERM pour un kill par défaut).

Exercice 4 (optionnel) : le seul paramètre actuel du programme est la racine du serveur web. Modifiez-le de façon à pouvoir lire ce nom de répertoire dans un fichier de configuration. Programmez une solution qui permet de relire cette racine quand le serveur reçoit le signal SIGUSR1.

Exercice 5 : on veut maintenant gérer des pages web dynamiques en PHP. Pour ce faire, nous allons utiliser une des fonctions exec* pour appeler l'interpréteur PHP en mode CGI (voir ci-dessous). Pour récupérer le résultat de l'exécution de PHP nous utiliserons les tubes de communication de façon à ce que la sortie standard de PHP soit redirigée dans un tube qui sera lisible par le serveur web. Attention : pour que ça fonctionne correctement il faudra calculer l'entête content-length correctement (elle ne doit pas compter la longueur des entêtes générées par PHP).

Exercice 6 (optionnel) : pour que PHP puisse gérer les paramètres des requêtes GET, que doit-on modifier ?

Annexe : Quelques informations sur CGI (Common Gateway Interface)…
But du CGI : faire communiquer un serveur avec des programmes externes. Le serveur trouve et exécute un programme et retourne le résultat au navigateur.
L'interface prévoit :

- un système de communication serveur - application CGI (passe par l'entrée standard STDIN de l'application) ;
- un système de communication application CGI - serveur (passe par la sortie standard STDOUT de l'application) ;
- un système qui passe les erreurs de l'application CGI vers le serveur par le biais du STDERR (sortie d'erreur standard) ;
- des variables d'environnement.

Les variables d'environnement permettent au programme CGI d'accéder à des informations relatives au serveur, au client ou à la requête. Elles sont créées par le serveur web avant de lancer le programme CGI.

Exemples :

Serveur

SERVER_NAME	nom DNS ou adresse IP du serveur gérant le script.

Client

HTTP_COOKIE	cookie sur le client

Requête

REQUEST_METHOD	get, post…
QUERY_STRING	chaîne qui suit le ?
SCRIPT_FILENAME	nom du programme CGI
REDIRECT_STATUS	permet de savoir s'il y a eu une redirection avant l'appel du programme CGI

Pour les récupérer en C : char* getenv(char* variable) ;
Exemple : char *valeur = getenv("SERVER_NAME") ;

Attention, pour que php marche en mode CGI il faut utiliser php-cgi et préciser, au minimum, les deux variables d'environnement suivantes : SCRIPT_FILENAME et REDIRECT_STATUS (voir : http://php.net/manual/en/security.cgi-bin.php).