Université Grenoble Alpes
Licence d'Informatique L3

Introduction aux Systèmes et Réseaux

TP : Client-serveur avec *sockets*

Pour réaliser ce TP, il faut avoir avoir lu les documents *"La bibliothèque d'entrée-sortie* Rio *"* et : *"Une bibliothèque C pour les* sockets *"*, disponibles sur Moodle.

Les programmes d'un système simple client-serveur utilisant les *sockets* en mode connecté sont donnés dans le placard : `echoclient.c`, `echoserveri.c`, `echo.c`.

1 Serveur itératif ECHO

Examiner les programmes ci-dessus pour bien comprendre leur fonctionnement, en vous aidant au besoin des documents techniques mentionnés ci-dessus.

Compiler séparément la partie serveur et la partie client. Lancer le serveur comme travail de fond à partir d'un `shell`, dans une fenêtre. Dans une autre fenêtre, lancer le programme client en premier plan, avec les paramètres adéquats. Observer le fonctionnement. Que se passe-t-il si vous lancez le client avant le serveur ? Que se passe-t-il si vous essayez d'accéder au même serveur à partir d'un processus client différent, pendant qu'un client est déjà connecté au serveur ?

Refaire les expériences ci-dessus en lançant le client et le serveur sur deux machines différentes.

Remarque importante Si vous êtes sur un serveur (mandelbrot), attention à ne pas tous utiliser le même numéro de port pour le serveur.

À faire en dehors de la séance de TP : Modifier les programmes pour afficher (côté client et côté serveur) les numéros des ports utilisés par les `sockets` à chaque extrémité de la connexion. Utiliser pour cela les primitives `getsockname` et `getpeername` décrites dans le document technique sur les *sockets*. Faire le lien entre ces informations et celles affichées par un utilitaire tel que `netstat` ou `ss`.

2 Serveur concurrent ECHO

On demande maintenant de transformer le serveur itératif en serveur concurrent (multiprogrammé). Un processus veilleur attend les requêtes des clients. Lorsqu'une requête arrive, le veilleur la fait traiter par un exécutant, puis se remet en attente. Il y a deux méthodes pour gérer les exécutants qui sont considérées dans les deux sections suivantes.

2.1 Création dynamique de processus

Une première méthode consiste, pour le veilleur, à créer un nouveau processus exécutant (par `Fork()`) pour servir chaque nouvelle demande de connexion.

Noter que les exécutants sont lancés en travail de fond, et passent à l'état zombi quand ils se terminent. Prévoir leur élimination par le veilleur (utiliser le traitant du signal `SIGCHLD`, puisque le veilleur ne peut pas se bloquer en attendant la fin des exécutants).

2.2 Pool de processus

L'opération `Fork()` est assez longue et peut représenter une fraction importante du temps de traitement de la requête si cette dernière est brève. Une méthode évitant cet inconvénient consiste à créer à l'avance un "pool" d'un nombre fixe `NPROC` de processus exécutants. Lorsqu'une requête arrive, elle est traitée par un de ces processus, qui se remet ensuite en attente. Avec cette méthode, on peut servir au maximum `NPROC` requêtes simultanément.

Réfléchir à la manière de faire attendre et d'activer les exécutants. Il y a deux classes de solutions, qui se distinguent notamment par le fait que la primitive `accept` est exécutée par le processus veilleur ou par les processus exécutants [1]. Choisir la solution la plus simple à réaliser (et justifier votre choix).

Expérimenter les programmes réalisés, avec création dynamique ou pool de processus, avec plusieurs clients simultanés. Afficher toujours les numéros des ports utilisés par les connexions.

Remarque : Veiller à supprimer tous les processus exécutants lors de l'arrêt du serveur. Si le serveur est lancé en tâche de premier plan, l'envoi d'un signal `SIGINT` via `Ctrl-C` sera automatiquement propagé à tous les processus du groupe de premier plan. En revanche, si le serveur est lancé en tâche d'arrière plan, la solution la plus simple consiste à envoyer un signal `SIGINT` (via la commande `kill`) au processus serveur initial (le veilleur) et à charger ce processus de retransmettre le signal à chacun de ses fils (via la programmation d'un traitant de signal approprié).

3 Transfert de fichier

Ce travail prépare le deuxième gros TP à rendre qui portera sur l'implémentation d'un serveur de fichiers à la FTP [2].

Remplacer le programme `echo` sur le serveur par un programme qui lit un nom de fichier et renvoie le contenu de ce fichier. Le client indique le nom du fichier demandé ; le serveur envoie en retour le contenu du fichier puis ferme la connexion (une seule demande de fichier par connexion).

Adapter en conséquence les programmes du client et du serveur. Faire fonctionner le système client-serveur avec ce nouveau service.

1. Noter que si plusieurs processus attendent sur un `accept` sur la même *socket* serveur, un seul processus obtient la connexion lors de l'arrivée d'une demande de la part d'un client (`connect`). Dans certaines versions/configurations du système, il est possible que tous les processus bloqués (sur un `accept` concernant la même *socket*) soient réveillés, mais un seul processus obtient la connexion (c'est-à-dire un numéro de descripteur valide).
2. File Transfer Protocol