Labo 3

Serveur concurrent avec connexion

© 2008 HEIG-VD, IICT

**Auteur**: J. Ehrensberger   
Version 1.0, Mai 2008

Objectifs du laboratoire

Ce laboratoire est le troisième d’une série de manipulations dans lesquelles nous allons développer une application de chat client – serveur.

Lors du premier labo nous avons développé un serveur itératif avec connexion pour une application de chat. Le défaut principal de ce serveur pour une application de Chat est qu’il ne gère qu’un client à la fois. Il ne permet donc pas de réaliser une communication entre plusieurs clients.

Dans ce troisième labo nous allons réaliser un serveur de chat concurrent avec connexion. Ce serveur sera capable de gérer plusieurs connexions à la fois. Cependant, comme chaque connexion sera génée par un processus esclave différent, la communication entre les clients ne sera pas possible. Le serveur à développer sera donc un seveur écho multi-client.

Après cette manipulation vous saurez :

1. Développer une application multi-processus (création de processus, terminaison de processus, gestion de processus)
2. Développer un serveur concurrent avec connexion, comme p.ex. un serveur Web ou FTP.

# Introduction

Le serveur développé lors du premier labo est du type itératif avec connexion. Il gère un seul client à la fois. Dans cette manipulation nous allons modifier ce serveur itératif ainsi de le convertir en un serveur multi-processus. Le processus maître reçoit les demandes de connexion et crée un nouveau processus esclave pour chaque client. Le processus esclave travail comme serveur écho : il reçoit les messages du client et les renvoie sans modifications.

Pour effectuer ce laboratoire, vous avez besoin de :

1. un ordinateur avec Linux, Mac OSX ou BSD
2. un éditeur de programme à votre choix
3. un compilateur C/C++ avec éditeurs de liens
4. l’outil make
5. les pages man pour le développement
6. l’outil netcat pour effectuer des tests

# Comportement du serveur itératif

Avant de commencer la réalisation du serveur concurrent nous allons analyser plus en détail le comportement d’un serveur itératif. L’objectif est de comprendre le fonctionnement de la file d’attente créée par le serveur avec l’appel listen.

## La file d’attente des connexions

Tâches à effectuer

* Utiliser le serveur TCP itératif réalisé lors du premier laboratoire.
* Vérifier que le serveur ne ferme pas le socket passif après avoir accepté une nouvelle connexion.
* Configurer une file d’attente de 5 places dans l’appel listen.
* Lancer Wireshark pour observer le trafic entre le serveur et les clients.
* Lancer le serveur.
* Se connecter au serveur avec un premier client et taper quelques messages. Ne pas fermer le client
* Se connecter au serveur avec un deuxième client et taper quelques messages.

A inclure dans le rapport

* Dessinez un diagramme de flèches qui montre le comportement du serveur lors de la connexion du deuxième client.
* Répondre aux questions :
  + Quel est le comportement observé du serveur lors de la connexion du deuxième client ? Expliquer ce comportement.
  + Que fait le serveur lorsqu’il reçoit des données du deuxième client ? Est-ce que les données sont acquittées ? Expliquer ce comportement.

## Gestion des connexions en attente

Maintenant nous aimerions savoir ce qui se passe lorsque beaucoup de clients se connectent à un serveur avec une file d’attente courte. Vérifiez que le serveur appelle la fonction listen avec 5 places dans la file d’attente.

Tâches à effectuer

* Relancer le serveur TCP itératif.
* Observer le trafic avec Wireshark.
* Lancer 6 connexions en parallèle avec NC. Ne pas taper un message.
* Continuer à créer de nouvelles connexions, tout en observant le trafic avec Wireshark.

A inclure dans le rapport

* Répondre aux questions :
  + A partir de combien de connexion observez-vous un comportement différent.
  + Quel comportement observez vous ?
  + Expliquer ce comportement.
  + Conclusion : quelle est la fonction de la file d’attente ?

# Serveur concurrent avec connexion

Après avoir étudié en détail le comportement du serveur itératif nous allons maintenant développer un serveur concurrent avec connexion. Le serveur à développer utilisera

* un processus maître pour accepter les nouvelles connexions
* un processus esclave par connexion.

Les processus esclaves ne font que renvoyer l’écho du message reçu au client connecté. Comme les processus sont complètement isolés, aucune communication entre les clients n’est possible.

Algorithme à réaliser

Les serveurs maître est esclaves réaliseront le comportement suivant :

|  |
| --- |
| **Processus maître** |
| 1. Créer un socket du type Stream (socket). |
| 1. Lier le socket à un port local (bind). |
| 1. Mettre le socket en mode passif (listen). |
| 1. Boucle infinie |
| 1. Recevoir une nouvelle requête (accept). |
| 1. Créer un nouveau processus esclave (*fork*). |
|  |
| **Processus esclave** |
| 1. Fermer le socket passif (close). |
| 1. Utiliser le nouveau socket créé par accept. |
| 1. Communiquer avec le client (read/recv et write/send). |
| 1. Fermer le socket (shutdown et close). |
| 1. Terminer. |

Tâches à effectuer

* Faire une copie des fichiers du programme du serveur TCP itératif.
* Modifier la méthode ChatServer::Accept :
  + Cette méthode réalise le comportement du processus maître. Elle contient une boucle infinie dans laquelle le processus appelle la fonction accept. Quand une nouvelle connexion est acceptée il crée un nouveau processus esclave qui gère cette connexion. Puis le processus maître ferme le nouveau socket pour lui et revient à l’appel d’accept.
  + Le processus esclave ferme le socket passif du processus maître. Puis il sort de la boucle du processus maître et appelle la méthode ChatServeur::MainLoop.
* Modifier la méthode MainLoop :
  + Cette méthode réalise le comportement du processus esclave.
  + Dans une boucle :
    - Lire un message et renvoyer l’écho.
    - Si la connexion est fermée par le client, terminer le processus avec exit(0).
* Modifier le programme principal (main()) :
  + Enlever l’appel de ChatServer::MainLoop du programme principal.
* Gestionnaire du signal SIGCHLD :
  + Ajouter une fonction C qui gère le signal SIGCHLD qui est envoyé au processus maître quand un processus esclave termine. Ce gestionnaire doit gérer la terminaison du processus esclave.

A ajouter au rapport

* Le programme complet (fichiers \*.cpp et \*.h) du serveur concurrent développé.
* Le protocole des tests effectués pour vérifier le bon fonctionnement du serveur.