



Rémy Malgouyres  
Université Clermont 1  
<http://laic.u-clermont1.fr/~mr/>

## Programmation système et réseaux

# TP n° 5 Système client serveur *TCP/IP*

**A rendre impérativement pour le lundi 5 novembre 9h**

### Objectifs :

Le but de ce TP-DM est d'implémenter une application avec système client-serveur avec connexion par sockets *TCP/IP*. On utilisera le client `telnet`.

## 1 Les fonctionnalité de l'application

Le but de ce *TP* est de créer une application client-serveur pour la gestion d'une base de données. Le but est de voir comment, en combinant les sockets et les threads avec mutex, on peut faire un système client-serveur avec accès concurrent aux données. Dans ce modèle, les données sont partagées, c'est à dire que tous les clients partagent les mêmes données. Comme plusieurs clients peuvent se connecter en même temps et chercher à modifier les données, on utilise des mutex pour éviter les conflits et l'incohérence des données.

**Exercice 1** Le but de l'exercice est de présenter un serveur gestionnaire d'une base de données de produits d'une entreprise.

- a) Définir une structure produit contenant un prix, un numéro de produit, et un nom. Définir aussi une structure commande (correspondant à une commande d'un client) contenant un tableau de numéros de produits (de la commande) et pour chaque produit des quantités commandées. Pour simplifier les problème, on ne gère pas les stocks de produits disponibles.
- b) Pour simplifier les problème, on suppose que les données de produit sont chargées en mémoire centrale par le programme serveur. Écrire une fonction de chargement d'un fichiers de produits
- c) Écrire une fonction de saisie d'une commande client par une socket. On utilisera les fonctions `read` et `write` pour dialoguer avec les client connecté par `telnet`. l'identificateur de socket est passé en paramètre.
- d) Écrire une fonction de sauvegarde d'une commande dans un fichier. On ouvrira le fichier (texte) en mode ajout et on écrira les données de la commande.
- e) Écrire une fonction qui affiche les données d'une commande pour le client. Le client est connecté par une socket dont l'identifiant est passé en paramètre.

f) Écrire une fonction `traite_connexion` qui prend en paramètre une socket, saisit une commande client via cette socket, affiche les données et le prix de la commande pour le client, et sauvegarde la commande dans un fichier `fichier_commande.txt`. L'écriture dans les fichier est protégé par un mutex pour le cas où eux clients saisiraient une commande au même instant.

g) Dans le `main`, on va créer un tableau de threads. Chaque thread aura son numéro et traitera une connexion de client avec sa socket.

```
// types, constantes et variables globales

#define NB_THREAD_MAX 10

pthread_mutex_t mutex_thread_occupe = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_ecrit_com = PTHREAD_MUTEX_INITIALIZER;

// donnée passée en paramètre à un thread
typedef struct

    int num_thread;
    int sock_connectee;
DataThread;

// Indique pour chaque thread s'il est occupé par une connexion
int thread_occupe[NB_THREAD_MAX];

// programme principal :

int main(void)
{
    pthread_t thread[NB_THREAD_MAX];
    DataThread data[NB_THREAD_MAX];
    [...]
}
```

Écrire une fonction de thread qui prend en paramètre l'adresse d'une structure `DataThread` :

- Un identifiant de socket par laquelle le thread dialogue avec un client;
- Un numéro de thread.

Le thread appelle la fonction `traite_connexion`, puis, avant de se terminer, met à 0 l'élément correspondant à son numéro dans le tableau `thread_occupe`. Le tableau `thread_occupe` est une donnée globale protégée par un mutex.

h) Écrire le programme principal qui boucle indéfiniment et attend une connexion d'un client par `accept`. A chaque connexion, le serveur crée un thread pour traiter la connexion. Pour cela, le serveur recherche un thread libre (valeur 0 dans le tableau `thread_occupe`) et appelle `pthread_create`. Le programme passe en paramètre l'adresse d'un `DataThread` pour le thread.