# Travail Pratique 2 (LINUX)
# Travail pouvant être effectué en équipe de 1, 2, 3 ou 4 personnes
# Vitual Machine System (VMS) et ressources partagées

En vous **inspirant** en premier lieu des notions expérimentées dans le programme permettant la gestion d'un VMS du TP1 (threads et sémaphores/mutex) et de l'utilisation des fenêtres (ex : ncurses/Qt) et des concepts de communication entre threads présentés en classe, vous devez implémenter un programme permettant la gestion concurrente d'un VMS similaire à celui développé au TP1, mais dans le TP2, à l'intérieur d'une architecture Client/Serveur. Des clients pourront alors acheminer des requêtes au serveur par une FIFO de transmission de transactions, le serveur (fonction readTrans() adaptée au niveau de la lecture des transactions) exécute chacune des transactions (threads) et retourne aux clients concernés par une FIFO assignée à chaque client l'information à afficher lors des transactions de (L) listages ou d'exécution (X). Ce(s) programme(s) (**un programme Serveur et un Client**) devrait comporter les spécifications suivantes:

a) Le programme **côté client** lors du démarrage ouvre deux fenêtres, une identifiée **TRANSMISSION**, l'autre **RECEPTION**. Le programme **côté client** pourra alors, par la fenêtre de TRANSMISSION, introduire des requêtes permettant d'effectuer des opérations sur une VM noVM donnée. Le programme client achemine ensuite, par exemple, des requêtes d'ajout de VM (transaction A) au serveur et ce par la FIFO de transmission de transactions.
b) Lors de la fermeture (en tapant quit (ou q) dans la fenêtre de transmission) de **l'application Client**, les fenêtres côté client sont fermées.
c) La FIFO de transmission de transactions devra s'appeler **FIFO_TRANSACTIONS**. Cette FIFO est créée par le programme **serveur** lors de son démarrage et est ouverte par le serveur en lecture (bloquante).
d) Ensuite, le **serveur (fonction *readTrans()*)** boucle en lecture sur la FIFO FIFO_TRANSACTIONS pour lire les informations provenant des clients. À chaque lecture, le **serveur** lit une structure **Info_FIFO_Transaction** ayant la forme suivante :

```
struct Info_FIFO_Transaction {
        int             pid_client;
        char            transaction[200];
}
```

Après la lecture d'une structure **Info_FIFO_Transaction** provenant d'un **client (identifié par son pid)**, le **serveur** démarre un thread qui traite chaque transaction provenant des clients. Le format des transactions (A : ajout d'une VM, L : listage des informations d'une VM, E : suppression d'une VM, X : pour l'exécution (interptétation) du code binaire contenue dans le fichier .olc3 dont le nom est spécifié comme argument) reçues est adapté au TP2 , par exemple :

```
A
L 1-4
A
L 1-4
```

A
L 1-4
E 2
L 1-4
A
L 1-4
X 1 Mult-Numbers.olc3
X 2 Mult-Numbers.olc3

e) Les threads du **côté serveur** qui traitent chaque requête des clients doivent alors ouvrir une FIFO en écriture (bloquante) pour permettre de répondre **(ex : optionnellement pour un ack de terminaison de transaction)** à chaque client, cette FIFO étant créée au préalable par le client lors de son démarrage. Cette FIFO a un nom correspondant à la concaténation de la chaîne de caractères "FIFO" concaténée au pid du thread côté client, par exemple, voir le code C :

```
struct Info_FIFO_Transaction    transaction;
int                             fifo_transaction_fd, client_fifo_fd;
char                            client_fifo[100];

fifo_transaction_fd = open("./FIFO_ TRANSACTIONS" , O_RDONLY);
 read(fifo_transaction_fd, &transaction, sizeof(transaction));
sprintf(client_fifo, "FIFO%d", transaction.pid_client));

client_fifo_fd = open(client_fifo,O_WRONLY);
```

f) Les threads du **côté serveur** doivent aussi répondre aux clients dans le cas d'une requête d'affichage (ex : L 1-4) ou les threads (côté **serveur**) déposent les structures **infoVM** sérialisées chacune dans une chaîne de caractères (dimension 400) déposées ensuite dans la FIFO (client_fifo) reliant le serveur et un client particulier qui sont à afficher du côté client.

g) Les threads du **côté serveur** doivent aussi répondre aux clients dans le cas d'une requête d'exécution (ex : X 1 Mult-Numbers.olc3) ou les threads (côté **serveur**) déposent les affichages produits par la fonction threadée ***executeFile()*** sérialisées dans une chaîne de caractères (dimension 400) déposées ensuite dans la FIFO (client_fifo) reliant le serveur et un client particulier, qui sont à afficher du côté client.

h) Les threads du côté **serveur** doivent aussi dans le cas de transactions de transmission d'information **(ex : ack de terminaison de transaction)** transmettre le message inséré d'abord dans une chaîne de caractères (dimension 400) et ensuite déposé dans la FIFO (client_fifo) reliant le serveur à chaque client récipiendaire du message particulier qui sont à afficher du côté client.

i) Quand un client décide de terminer la transmission de transactions il doit détruire la FIFO client_fifo (unlink()).

j) Du côté client les E/S doivent s'effectuer chacune à l'intérieur d'une fenêtre. L'introduction des transactions se fait donc de façon manuelle à l'intérieur d'une fenêtre du côté client (TRANSMISSION). Les réponses provenant du serveur sont affichées à

l'intérieur d'une autre fenêtre (RECEPTION) en caractères d'une couleur différente pour distinguer les réponses découlant de listage et celles de transmission de messages.

k) Du côté **client** vous devriez utiliser des threads de façon à ce que la lecture des transactions et l'affichage des réponses reçues du serveur soient indépendants.

**Contraintes de la partie 2 (10 points):**

Le rapport de ce travail doit être remis le **29 novembre 2022.** La correction du travail est basée sur les critères suivants :

**Fonctionnement des programmes** 50 %
**Programmes** (code du main(), codes des procédures, fichier en-tête, makefile)
**Style de programmation** 50 %
(documentation, efficacité de la solution, validation, originalité de la solution)

## Documents à remettre :

**Vous devez déposer un manuel d'utilisateur de votre application sur le portail de cours, incluant votre *Username, Password,* ainsi que le chemin pour accéder à votre dossier TP2, le nom de chaque fichier exécutable du client et du serveur.**
**Avis aux étudiants qui travaillent en VIRTUAL BOX, voir à me transmettre un rapport contenant les résultats de l'éxécution du fichier de transactions trans2.txt ainsi que le code en C des modules de code.**