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

En vous **inspirant** en premier lieu des notions expérimentées dans le programme permettant la gestion d'un VMS du TP1 (concurrence et exclusion mutuelle) et du TP2 (concepts de fenêtres et communication par FIFO dans une architecture Client/Serveur) et des concepts de communication par socket présentés lors dans les documents cour7sif1015_LINUX-FondBlanc.ppt (cour7sif1015_LINUX-FondBlanc.mp4) et cour8sif1015_LINUX.ppt (cour8sif1015_LINUX.mp4), vous devez implémenter un programme permettant la gestion concurrente d'un VMS aussi à l'intérieur d'une architecture Client/Serveur ***mais orientée Socket.*** **Des clients pourront alors acheminer des requêtes au serveur par un Socket de communication qui leur est à chacun dédié, le serveur devra gérer chaque client de façon concurrente, en exécutant chacune des transactions de ces clients et retournant des messages aux clients (ex : commandes de listage (L) ou d'exécution (X)) par le même Socket dédié à chaque client (connexion de communication). Ce(s) programme(s) (Client/Serveur) devraient comporter au moins les spécifications suivantes:**

a) Comme au TP2, 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 de gestion de VMS. Pour le TP3, le programme client devrait ensuite pouvoir acheminer, par exemple, des requêtes d'ajout (A) (ou L, E, X) au serveur **et ce par un socket de communication avec le serveur.**
b) Comme au TP2, lors de la fermeture (en tapant quit dans la fenêtre de transmission) de **l'application Client**, les fenêtres côté client sont fermées, mais dans le cas du TP3, le **socket de communication avec le serveur est aussi fermé**.
c) Lors du démarrage du serveur un socket est créé avec un numéro de port, spécifique aux services offerts par ce serveur, ce numéro de port qui avec l'adresse IP du serveur (ou nom de domaine) permettent aux clients d'identifier le serveur. Le serveur crée pour sa part un **socket listener** et pour chaque client, lors de la réception de requêtes de connexion, un socket de communication (voir l'appel système ***accept()***) pour permettre la communication avec le serveur.
d) Votre programme serveur doit gérer la connexion et la déconnexion des clients.
e) Le serveur doit créer un thread par connexion (par client) qui permet de traiter les requêtes reçues de chaque client (**voir fonction *readTrans()***).
f) Chaque thread créé côté serveur, reçoit les requêtes d'un client particulier, traite chaque requête individuellement et lors de listage (ou d'affichage des résultats d'exécution (transaction X)) redirige les informations demandées par ce client connecté, par le socket de communitation dédié à ce client.
g) Le serveur reçoit des structures Info_Transaction pouvant avoir la même forme qu'au TP2 suivante :

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

h) Chaque requête reçue par le serveur est traitée de façon **concurrente** donc chaque client sera associé à un socket qui permettra la communication entre le serveur et chaque client branché
i) Pour tester votre architecture Client/Serveur sur DMILinux (ou sur votre machine virtuelle) vous pouvez utiliser l'adresse IP 127.0.0.1.

Contraintes de la partie 2 (10 points):

Le rapport de ce travail doit être remis le **20 décembre 2022 avant 23h55.** 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 %
(modularité, documentation, indentation, utilisation de constantes, efficacité des programmes, validation, originalité)

Documents à remettre :

1) **Si votre application Client/Serveur est développée sur DMILINUX, veuillez déposer (un seul dépôt est requis par groupe) 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 TP3, le nom de chaque fichier exécutable du client et du serveur.**
2) **Si votre application Client/Serveur est développée sur Virtual Box, veuillez déposer (un seul dépôt est requis par groupe) un rapport présentant des affichages de votre application client, fenêtres TRANSMISSION et RECEPTION lors de transactions A E L et X. Veuillez aussi ajouter les sources des applications client et serveur.**