Calendrier Commun pour Kaamelott

C'en est trop ! Le roi Arthur, illustre roi de Bretagne, dans son royaume de Kaamelott n'arrive pas à organiser des sessions de table ronde. En effet les différents chevaliers de la table ronde sont toujours accaparés par leurs tâches/activités et impossible pour Arthur de trouver des créneaux pour faire des réunions autour de la fameuse table ronde.

Pour régler ce problème, vous êtes commissionné par le roi de produire une application de calendrier de groupe. L'idée est toute simple, il veut simplement une application qui regroupera le planning de ses différents chevaliers afin de pouvoir organiser des sessions de table rondes.

Description de l'application

L'application sera à disposition des chevaliers et d'Arthur. Tous seront priés de renseigner leur agenda via ces différentes commandes :

— `ADD Lancelot yoga lundi 15`
Ajoute pour l'utilisateur `lancelot` l'activité `yoga` pour le `lundi` à `15h`

— `DEL Perceval tricot dimanche 9`
Retire pour l'utilisateur `Perceval` l'activité `tricot` du `dimanche` à `9h`

— `SEE`
Affiche la totalité du planning

```
Mercredi 11h: Lancelot a footing
Lundi 17h: Caradoc a goûter
Mercredi 10h: Arthur a méditation
```

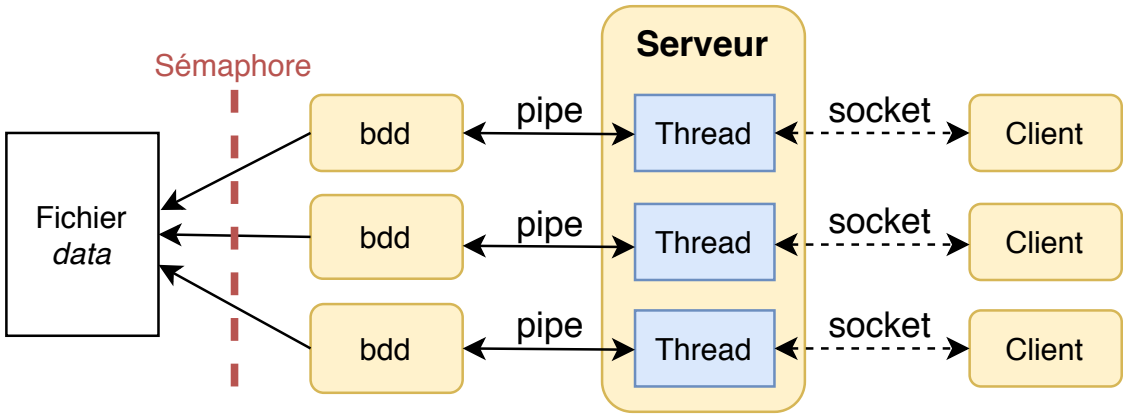

FIGURE 1 – Schéma des composants de l'application

1 Base de données des activités

Pour commencer nous allons écrire la partie base de donnée où notre application a plusieurs fonctions nous permettant de créer/éditer/lire les données stockées dans un fichier *data*. Pour cela ouvrez le fichier *bdd.c* et répondez aux questions suivantes.

A votre disposition Dans le fichier *bdd.h* vous sont procurés l'énumérateur `Day` correspondant aux jours de la semaine et la structure `Data` correspondant aux données stockées dans notre base de données. Des fonctions concernant l'utilisation de ces objets C vous sont fournis dans *bdd.c*. **Les objets C et les signatures fournies sont des suggestions pour l'exercice, vous être libres de les modifier à votre convenance.**

Question 1 *Écrivez une fonction* `add_data` *qui nous permet d'ajouter des données dans le fichier* ***data****.*
- *Si le fichier* ***data*** *n'existe pas, créez le fichier*
- *Chaque ligne de* ***data*** *correspond à une donnée avec un nom de chevalier, un nom d'activité, un jour et une heure*
- *Stockez les données sous la forme :* `nom,activité,jour,heure`

Question 2 *Écrivez une fonction* `delete_data` *qui nous permet de supprimer une donnée du fichier data.*
- *Si l'activité mentionnée n'est pas trouvée, signalez-le*
- *Note : vous aurez besoin de la fonction* `strcmp`*.*

Question 3 *Écrivez une fonction d'affichage* `see` *nous permettant d'afficher le planning complet dans le style de l'exemple présenté auparavant*

Question 4 *Complétez la fonction* `main` *afin que le programme puisse :*
- *recevoir en arguments des chaînes de caractères correspondant aux différentes commandes de l'application :* `ADD, DEL, SEE`*.*
- *applique les opérations de base de données correspondantes aux arguments reçus*
- *on s'attend à que ces différentes commandes fonctionnent :*

```
$ ./bdd ADD Lancelot padel lundi 18
$ ./bdd SEE
Lundi 18h: Lancelot a padel
$ ./bdd DEL Lancelot padel lundi 18
```

- *en cas d'erreur le programme doit retourner la valeur 1 et 0 si les opérations se sont déroulées correctement*

2 Serveur

Maintenant nous allons procéder à l'application serveur (*server.c*) qui reçoit les commandes telles que décrites au début du sujet. A la réception des commandes notre serveur appelle notre application *bdd* dans un nouveau processus.

Question 5 *Maintenant que nos fonctions de gestion des données sont prêtes on s'attaque à la partie gestion des commandes. Complétez la fonction* `main` *qui :*
- *doit être en attente de commandes*
- *crée un nouveau processus chargé d'effectuer les opérations sur notre base de données avec* `bdd` *(notre application)*

— *si les commandes* `ADD` *et* `DEL` *se sont bien déroulées, affichez "Commande effectuée"*

3 Connexion avec l'application client

Pour l'instant notre application ne gère les commandes que si on lui donne les commandes directement. Nous souhaitons proposer une application client aux chevaliers pour qu'ils puissent renseigner leur planning depuis leur propre ordinateur.

Question 6 *Dans* ***server.c*** *configurez et déployez une socket réseau qui gèrera les connexions entrantes.*
- *notre connexion utilisera le protocole TCP/IP*
- *nous utiliserons les adresses IPV4*

Note *: Des informations concernant les connexions sont stockés dans le fichier utils.h.*

Question 7 *L'application client est codée dans* ***client.c****. Ouvrez le fichier et programmez une socket se connectant à notre serveur et essayez de lui transmettre des messages.*

Question 8 *Écrivez une fonction* ***process_communication*** *dans server.c qui traitera les commandes reçues depuis le client.*
- *les commandes du client seront transmises à un processus fils d'une manière similaire au travail effectué auparavant (Question* **??***).*
- *les réponses de la base de données seront transférées au serveur par un pipe*
- *ces réponses seront ensuite retournées à l'application client par la socket réseau établies auparavant*

4 Multi-threading pour les performances !

Pour booster les performances de notre applications on va utiliser de la programmation concurrentielle avec des threads. Notre but est que les connexions entrantes sur notre serveur soient gérées par différents threads. Cependant pour ne pas compromettre l'état de notre base de données il va falloir sécuriser son accès.

Question 9 *On va commencer par protéger l'accès à notre base de données afin qu'elle ne soit manipulée que par un seul processus à la fois. Utilisez un sémaphore pour prévenir tout accès concurrentiel à notre base de données.*

Question 10 *Modifiez le code du serveur afin qu'il utilise plusieurs threads pour gérer les connexions entrantes.*

5 Bonus

Si il vous reste plein de temps vous pouvez améliorer l'application voici des suggestions :

Question 11 *Utilisez **Valgrind** et corrigez toutes les fuites mémoires potentielles. Pour plus d'informations sur Valgrind allez voir l'annexe technique.*

Question 12 *Optimisons notre application en faisant meilleur usage de la programmation concurrentielle. Plutôt que que d'avoir un unique accès à notre base de donnée appliquez le principe **un unique écrivain OU plusieurs lecteurs** sur notre base de données.*

Question 13 *Ajout des commandes `SEE_DAY` et `SEE_USER` présentées ci-dessous.*

— `SEE_DAY mardi`
Affiche le planning de la journée de `mardi`

```
Planning du Lundi:
13h: Arthur a pétanque
10h: Caradoc a raclette
18h: Perceval a piscine
```

— `SEE_USER Bohort`
Affiche le planning de l'utilisateur `username`

```
Planning de Bohort:
Jeudi 20h: Pourfendage de bandits
Mardi 08h: Cours à CentraleSupélec
```

Question 14 *Modifiez votre code afin d'ajouter ces différentes fonctionnalités :*
- *affichage des plannings par ordre chronologique*
- *un utilisateur ne peut avoir plusieurs activités au même moment*
- *toutes les commandes passées à la base de données sont journalisées dans un fichier `data.log`*