Exercice à rendre 2

On désire simuler le fonctionnement d'un marché où des consommateurs (ou clients) vont acheter leurs provisions chez différents commerçants (ou vendeurs). L'étal d'un vendeur est représenté par un fichier contenant la quantité de produits disponibles (on suppose que chaque vendeur ne vend qu'un seul produit). Si un produit n'est plus disponible sur l'étal, le vendeur peut puiser dans la réserve (que l'on suppose infinie) située dans sa camionnette pour reconstituer son étal.

Pour simuler le marché, on demande de rédiger les programmes suivants :

— `vendeur` *`produit quantité`*
Ce programme simule un commerçant qui installe son étal avec la quantité indiquée de produits ou qui ajoute les produits si son étal est déjà ouvert. Concrètement, il agit sur le fichier nommé `produit`. Si des clients étaient en attente pour acheter ce produit, ils doivent être rudement contents. Une fois les produits amenés sur l'étal, le programme `vendeur` peut s'arrêter. On peut à tout moment le relancer pour ajouter de nouveaux produits.
Lorsque le commerçant souhaite fermer son étal et partir, il doit appeler le programme avec une quantité nulle. L'étal est dès lors fermé, même s'il restait des produits non vendus. Les clients en attente doivent être réveillés, ils constatent que l'étal est fermé et rentrent chez eux, dépités.

— `client` $produit_1$ $quantité_1$ `...` $produit_n$ $quantité_n$
Ce programme simule un client avec sa liste de courses : il doit acheter les produits indiqués. Si un produit est inconnu, c'est-à-dire si le fichier correspondant est vide, le client râle et se termine (avec ou sans les produits déjà achetés, selon ce qui vous semblera le plus simple). Si un produit n'est pas disponible, le client attend que le commerçant réapprovisionne son étal. Si le commerçant ferme son étal, le client râle et arrête là ses courses.

Par exemple :

```
> ./vendeur banane 3
# le vendeur de bananes en dispose 3 sur son étal
> ./vendeur pomme 1
# le vendeur de pommes en dispose 1 sur son étal
```

```
> ./client pomme 1 banane 2
# ok pour acheter 1 pomme et 2 bananes
> ./client pomme 1
# ah... plus de pommes disponibles
# ... on attend
# Ça y est ! on peut les avoir
>
```

```
> ./client banane 2
# plus assez de bananes
# ... on attend
# ... on attend
# aïe ! le vendeur ferme !
Erreur : vendeur a ferme
```

```
> ./vendeur pomme 2      # on ajoute 2
> ./vendeur banane 0     # on ferme !
> ./vendeur pomme 0      # on ferme aussi !
```

Cette simulation doit être réalisée à l'aide de fichiers et de sémaphores POSIX nommés pour la synchronisation, à l'exclusion de tout autre mécanisme. Vous pouvez ajouter des informations supplémentaires aux fichiers selon vos besoins, mais vous décrirez dans un commentaire dans le fichier `lib.h` les informations que vous ajoutez, leur structure et leur utilité.

Pour simplifier, on supposera que les noms de produits sont des noms simples dans le répertoire courant (pas de sous-répertoires ou de chemins compliqués) et qu'ils ont une taille limitée à 10 octets. Vous pourrez utiliser des fonctions de bibliothèque si vous le souhaitez, y compris pour l'accès aux fichiers. Aucune attente active ne sera tolérée, même ralentie.

Un script de test est mis à votre disposition sur Moodle. Celui-ci exécute votre programme sur des jeux de tests qui serviront de base à l'évaluation de votre rendu. La commande suivante permet de lancer les tests : `sh ./test2.sh`

N'hésitez pas à contacter votre enseignant si vous constatez un comportement anormal ou si vous souhaitez ajouter un test.

Pour faciliter l'évaluation de votre travail, il est impératif d'utiliser le répertoire qui vous est fourni sur Moodle. Vous devrez rédiger les fichiers `.c` correspondant aux programmes demandés, ainsi que les fichiers `lib.c` et `lib.h` contenant les définitions communes utilisées dans les deux programmes. Pour construire l'archive à déposer sur Moodle, vous utiliserez impérativement la cible `devoir.tgz` du `Makefile`.

Cet exercice à rendre est **individuel**. On rappelle que la copie ou le plagiat sont sévèrement sanctionnés.