Gestion de stock

On souhaite modéliser un mécanisme simplifié de gestion de stock pour le commerce en ligne à l'aide de processus (*pas* de threads) représentant les différents clients. Le stock est conservé en mémoire partagée, sous forme d'une liste d'articles simplement chaînée pour simplifier. Chaque article du stock est représenté par sa référence (un entier strictement positif) et la quantité disponible. Les quantités d'articles ne seront pas représentées comme des valeurs entières dans la liste : on utilisera à la place des sémaphores (un par référence).

La liste simplement chaînée ne pouvant être réalisée à l'aide de pointeurs, on considérera le segment de mémoire partagée comme un tableau : les « pointeurs » sont des indices dans ce tableau et la valeur `NULLPTR` (en pratique, -1) indique le pointeur invalide. Dans l'exemple ci-après, où le champ `suiv` réalise le chaînage, les cellules occupées sont 1, 5 et 3 (dans cet ordre) et les cellules libres sont 0, 2, 6 et 4 :

	0	1	2	3	4	5	6
occupes: 1	suiv: 2	suiv: 5	suiv: 6	suiv: ⊥	suiv: ⊥	suiv: 3	suiv: 4
libres: 0	ref: ×	ref: 17	ref: ×	ref: 42	ref: ×	ref: 68	ref: ×
	sem: ...	sem: ...	sem: ...	sem: ...	sem: ...	sem: ...	sem: ...

On demande d'écrire, en utilisant exclusivement l'API POSIX, les programmes (ou variantes) ci-dessous. Les sources des programmes sont fournis, il n'est pas nécessaire de les modifier. En revanche, ces sources appellent les fonctions du fichier `stock.c` que vous devrez écrire (avant de commencer, lisez attentivement les commentaires et le code des fichiers `stock.h` et `stock.c`). Dans chaque cas, le nom des fonctions à écrire est précisé.

1. `ctrl init` *`nref`* — Fonctions : `stock_create()`, `stock_unmap()`

   Cet appel du programme `ctrl` initialise le segment de mémoire partagée et les mécanismes de synchronisation associés, pour un maximum de *`nref`* références d'articles. Au début, aucun article n'est dans le stock, l'ensemble des articles doit donc être placé dans une liste d'articles « libres ».
2. `ctrl destroy` — Fonction : `stock_destroy()`

   Cet appel supprime (ou plutôt demande la suppression) du segment de mémoire partagée.
3. `ctrl dump` — Fonction : `stock_map()`

   Cet appel affiche diverses tailles, ainsi que le contenu brut du segment de mémoire partagée. La seule fonction à écrire ici projette un segment existant dans l'espace d'adressage du processus. La fonction `stock_dump()` vous est fournie. Utilisez cet appel de `ctrl` pour vérifier que votre segment de mémoire partagée est bien initialisé (et bien maintenu correct par les appels ci-dessous).
4. `aref` *`réf`* [*`réf`* ...] — Fonction : `stock_addref()`

   Ce programme est utilisé pour référencer un nouvel article (i.e. ajouter une nouvelle référence dans le stock). Si la référence existe déjà, un simple message d'erreur est affiché. Sinon, la référence est insérée dans la liste avec une quantité nulle. Si le nombre maximum de références est dépassé, on se contentera d'afficher une erreur.
5. `rref` *`réf`* [*`réf`* ...] — Fonction : `stock_remref()`

   Ce programme est utilisé pour déréférencer un article existant (i.e. retirer une référence du stock). Si la référence n'existe pas, un simple message d'erreur est affiché. Sinon, la référence est retirée de la liste et le sémaphore associé correctement détruit.

Le site de commerce en ligne devant accueillir un grand nombre de clients, vous devrez dans la suite placer vos synchronisations de manière à profiter au maximum du parallélisme offert par le matériel. Par exemple, si deux clients sont en cours d'achat (retrait d'articles), ils doivent pouvoir faire les traitements en parallèle. De même, une livraison doit pouvoir s'effectuer alors qu'un achat est en cours.

6. `ajout [--slow]` *`réf qté`* [*`réf qté`* ...] — Fonction : `stock_add()`

   Ce programme, appelé par exemple lors d'une livraison, ajoute dans le stock le ou les articles dont la référence et la quantité sont indiquées en argument. Les références doivent avoir été ajoutées au préalable, sinon un message est affiché et la référence est ignorée.

7. `retrait [--slow] réf qté [ réf qté ... ]` Fonction : `stock_rem()`

   Ce programme est appelé par exemple lors d'un achat par un client : les articles demandés sont retirés du stock s'ils sont disponibles. Sinon, le programme attend que le stock se remplisse. Si une référence n'existe pas, on se contentera d'afficher une erreur et d'ignorer la référence.

Notez que ce dernier programme risque de causer des interblocages. Essayez d'en provoquer un (l'option `--slow` est là pour vous aider). Notez que vous pouvez toujours résoudre un interblocage en exécutant `ajout` dans un autre terminal. On ne vous demande pas de pallier cet interblocage.

8. `liste` Fonction : `stock_list()`

   Ce programme dresse l'inventaire du stock : pour chaque référence, la quantité disponible est affichée.