Examen SEM

L'objectif est de simuler un réseau de stations Vélib. A la fin de l'examen, chacun doit envoyer [1] un mail [2] avec en fichier attaché un répertoire [3] nommé avec prénom et nom et contenant pour chaque exercices les codes en C des programmes [4]

Dans chaque exercice, il est important que :

— Toute fonction susceptible d'émettre un jeton récupérable par la fonction `GetLastError()` doit être testée.
— La gestion de l'héritage des Handle doit être bien faite.
— Tous les Handle créés doivent être fermés dès qu'ils ne sont plus utilisés. Il convient cependant de prendre en compte qu'un Handle fermé ne peut plus être hérité et qu'un objet dont à un instant donné tous les handle associés sont fermés est un objet qui n'existe plus et qui ne peut donc plus être ouvert sans être à nouveau créé.

Exercice 1 *Écrivez un programme appelé* `centre.c` *qui génère 3 processus exécutant chacun le programme appelé* `station.c`. *Chaque nouveau processus indique dans un nouvelle fenêtre son numéro de station. Ces deux programmes sont à transmettre sous le nom de* `centre1.c` *et* `station1.c`.

Exercice 2 *Modifiez ces deux programmes de façon que le processus associé à* `centre.c` *se termine après que l'on appuie sur la touche Entrée (utilisation de* `getchar()`*) et la fin de ce processus met fin à chacun des processus associé à* `station.c`. *Ces deux programmes sont à transmettre sous le nom de* `centre2.c` *et* `station2.c`.

Exercice 3 *Modifiez ces deux programmes de façon que le processus associé à* `centre.c` *crée 3 canaux de communications (pipe) non-nommés. Chacun de ces canaux est ensuite utilisé par ce processus pour transmettre à chaque station le message "bonjour station 1", "bonjour station 2", "bonjour station 3". Ces messages sont affichés à leur réception par chaque processus assocé à* `station.c`. *Ces deux programmes sont à transmettre sous le nom de* `centre3.c` *et* `station3.c`.

On définit maintenant un nouveau type avec

```
typedef enum {
  PRENDRE=0,
  RENDRE=1,
  FIN=2,
} ACTION;
```

On définit trois nouvelles fonctions :

```
VOID prendre_velib(INT numSta,HANDLE hPipe[3]);
VOID rendre_velib(INT numSta,HANDLE hPipe[3]);
VOID terminer(HANDLE hPipe[3]);
```

La première fonction transmet au canal approprié l'information selon laquelle un vélib est pris à la station `numSta`. La deuxième fonction transmet au canal approprié l'information selon laquelle un vélib est rendu à la station `numSta`. La troisième fonction transmet au canal approprié l'information selon laquelle la simulation est terminée et le processus informé n'attend plus une information sur le canal mais seulement la fin du processus associé à `centre.c`. `numSta` est un entier entre 1 et 3. Pour simplifier l'implémentation il est conseillé de transmettre dans le canal directement les octets associés au type énuméré (i.e. concrètement il convient de mettre une adresse).

Exercice 4 *Modifiez et rajoutez ces deux fonctions à* `centre.c` *de façon à modéliser le fait qu'un vélib de la station 1 est amenée à la station 2 et un vélib de la station 2 part. Modifiez* `station.c` *de façon à faire un affichage approprié. L'affichage du processus associé à la station 1 est :*

```
station numero 1
Un velib vient de quitter cette station.
```

L'affichage du processus associé à la station 2 est :

1. à l'adresse : gabriel.dauphin@univ-paris13.fr
2. Le mail doit contenir la liste des exercices réalisés et le prénom et le nom.
3. Le répertoire doit être nommé avec le format suivant `prenom_nom`.
4. `centre1.c` et `station1.c` pour le premier exercice, `centre2.c` et `station2.c` pour le deuxième exercice, `centre3.c` et `station3.c` pour le troisième exercice, `centre4.c` et `station4.c` pour le quatrième exercice, `centre5.c` et `station5.c` pour le cinquième exercice.

```
station numero 2
Un velib vient de revenir a cette station.
Un velib vient de quitter cette station.
```

L'affichage du processus associé à la station 3 est :

```
station numero 3
```

Les deux programmes `centre.c` *et* `station.c` *sont à transmettre sous le nom de* `centre4.c` *et* `station4.c`.

Exercice 5 *L'utilisation d'un vélib est modélisée par une unité d'exécution du processus associé à* `centre.c`. *Cette utilisation est caractérisée par un numéro de station de départ (entier de 1 à 3), une durée du trajet (entier entre 0 et 10) et un numéro de station d'arrivée (entier de 1 à 3). Ces informations sont tirées aléatoirement un peu avant la création de chaque unité d'exécution. Ces unités d'exécutions sont créés les unes après les autres avec un délai entre deux créations qui est tirée aléatoirement et correspond à un nombre entre 0 et 10 secondes. La création de ces unités d'exécution se termine quand l'utilisateur tape sur une touche. Le fait de taper Entrée met fin à la simulation. Pour simplifier l'implémentation, c'est la même plage mémoire qui est transmise à toutes les unités d'exécutions, mais chaque unité d'exécution lance un signal lorsqu'elle a fini de copier l'information transmise par cette plage mémoire et susceptible d'être modifiée. C'est seulement à la réception de ce signal qu'une unité d'exécution supplémentaire est créée.*

Voici un exemple d'affichage.

L'affichage du processus associé à `centre.c` *est*

```
velib quitte station 1 et dans 8 secondes arrivera a station 2.
velib quitte station 2 et dans 5 secondes arrivera a station 2.
velib quitte station 1 et dans 9 secondes arrivera a station 3.
```

L'affichage du processus associé à la première station est

```
station numero 1
Un velib vient de quitter cette station.
Un velib vient de quitter cette station.
```

L'affichage du processus associé à la deuxième station est

```
station numero 2
Un velib vient de quitter cette station.
Un velib vient de revenir a cette station.
Un velib vient de revenir a cette station.
```

L'affichage du processus associé à la troisième station est

```
station numero 3
Un velib vient de revenir a cette station.
```

Les deux programmes `centre.c` *et* `station.c` *sont à transmettre sous le nom de* `centre5.c` *et* `station5.c`.