Programmation système (3° partie) : Communication inter-processus

Thierry Vaira

BTS SN-IR Avignon

v0.1 16 février 2016

Sommaire

1. Les communications inter-processus
2. Les tubes
3. Les signaux
4. L'API WIN32 (Windows)

Frères des Écoles Chrétiennes
Saint-Jean-Baptiste de la Salle - Avignon

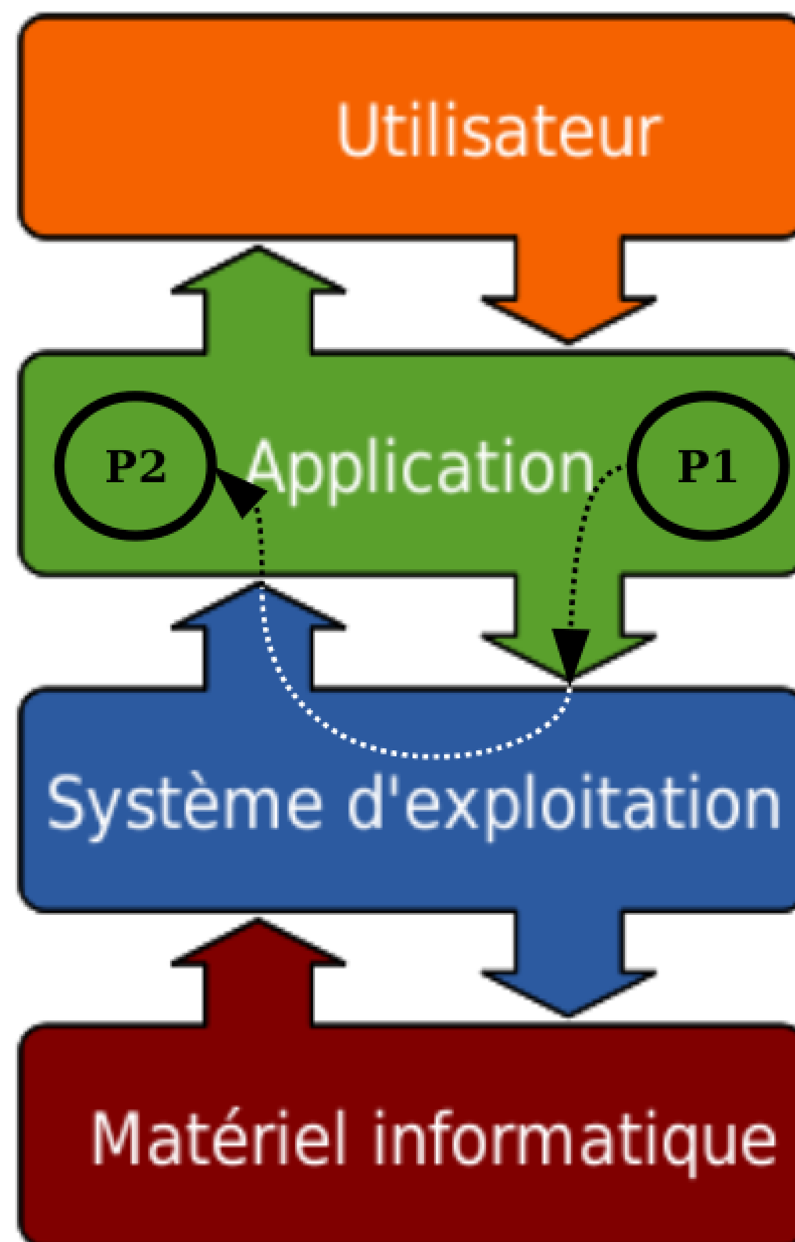
IPC

Les **communications inter-processus** (*Inter-Process Communication* ou **IPC**) regroupent un ensemble de mécanismes permettant à des processus concurrents (ou distants) de communiquer. Ces mécanismes peuvent être classés en trois catégories :

- les outils permettant aux processus de **s'échanger des données**
- les outils permettant de **synchroniser les processus**, notamment pour gérer le principe de section critique
- les outils offrant directement les caractéristiques des deux premiers (échanger des données et synchroniser des processus)

Frères des Écoles Chrétiennes
Saint-Jean-Baptiste de la Salle - Avignon

Principe



Frères des Écoles Chrétiennes
Saint-Jean-Baptiste de la Salle - Avignon

Échange de données

- Les **fichiers** peuvent être utilisés pour échanger des informations entre deux, ou plusieurs processus. Et par extension de la notion de fichiers, les **bases de données** peuvent aussi être utilisées pour échanger des informations entre deux, ou plusieurs processus.
- La **mémoire** (principale) d'un système peut aussi être utilisée pour des échanges de données. Suivant le type de processus, les outils utilisés ne sont pas les mêmes :
  - Dans le cas des processus lourds, l'espace mémoire du processus n'est pas partagé. On utilise alors des mécanismes de **mémoire partagée**, comme les **segments de mémoire partagée** pour Unix.
  - Dans le cas des processus légers (*threads*) l'espace mémoire est partagé, la mémoire peut donc être utilisée directement.
- Quelle que soit la méthode utilisée pour partager les données, ce type de communication pose le problème des **sections critiques**.

Synchronisation

Les mécanismes de synchronisation sont utilisés pour résoudre les problèmes de **sections critiques** et plus généralement pour bloquer et débloquer des processus suivant certaines conditions :

- Les **verrous** permettent de bloquer tout ou une partie d'un fichier
- Les **sémaphores** (et les **mutex**) sont un mécanisme plus général, ils ne sont pas associés à un type particulier de ressource et permettent de limiter l'accès concurrent à une section critique à un certain nombre de processus
- Les **signaux** (ou les **évènements**) permettent aux processus de communiquer entre eux : réveiller, arrêter ou avertir un processus d'un événement

L'utilisation des mécanismes de synchronisation est difficile et peut entraîner des **problèmes d'interblocage** (tous les processus sont bloqués).

Échange de données et synchronisation

Ces outils regroupent les possibilités des deux autres et sont souvent plus simples d'utilisation.

- L'idée est de communiquer en utilisant le principe des **files** (notion de boite aux lettres), les processus voulant envoyer des informations (**messages**) les placent dans la file ; ceux voulant les recevoir les récupèrent dans cette même file. Les opérations d'écriture et de lecture dans la file sont bloquantes et permettent donc la synchronisation.
- Ce principe est utilisé par :
  - les **files d'attente de message (*message queue*)** sous Unix,
  - les ***sockets* Unix ou Internet**,
  - les **tubes (nommés ou non)**, et
  - la transmission de **messages (*Message Passing*)** (DCOM, CORBA, SOAP, ...)

Frères des Écoles Chrétiennes
Saint-Jean-Baptiste de la Salle - Avignon

IPC Système V (Unix) / POSIX

Les IPC (Système V ou POSIX) recouvrent 3 mécanismes de communication entre processus (*Inter Processus Communication*) :

- Les **files de messages** (*message queue*), dans lesquelles un processus peut glisser des données ou en extraire. Les messages étant typés, il est possible de les lire dans un ordre différent de celui d'insertion, bien que par défaut la lecture se fasse suivant le principe de la file d'attente.
- Les **segments de mémoire partagée** (*shared memory*), qui sont accessibles simultanément par deux processus ou plus, avec éventuellement des restrictions telles que la lecture seule.
- Les **sémaphores**, qui permettent de synchroniser l'accès à des ressources partagées.

Les IPC permettent de faire communiquer deux processus d'une manière asynchrone, à l'inverse des tubes.

Files de messages

- Les **files de messages** sont des **listes chaînées** gérées par le noyau dans lesquelles un processus peut déposer des données (**messages**) ou en extraire. Elle correspond au concept de **boîte aux lettres**.
- Un **message** est une structure comportant un nombre entier (le type du message) et une suite d'octets de longueur arbitraire, représentant les données proprement dites.
- Les messages étant typés, il est possible de les lire dans un ordre différent de celui d'insertion. Le processus récepteur peut choisir de se mettre en attente soit sur le premier message disponible, soit sur le premier message d'un type donné.

Frères des Écoles Chrétiennes
Saint-Jean-Baptiste de la Salle - Avignon

Files de messages (suite)

Les **avantages** principaux de la file de message (par rapport aux tubes et aux tubes nommés) sont :

- un processus peut émettre des messages même si aucun processus n'est prêt à les recevoir
- les messages déposés sont conservés, même après la mort de l'émetteur, jusqu'à leur consommation ou la destruction de la file.

Le principal **inconvénient** de ce mécanisme est la limite de la taille des messages (8192 octets sous Linux) ainsi que celle de la file (16384 octets sous Linux).

Frères des Écoles Chrétiennes
Saint-Jean-Baptiste de la Salle - Avignon

Segment de mémoire partagée

Les processus peuvent avoir besoin de partager de l'information. Le système Unix fournit à cet effet un ensemble de routines permettant de créer et de gérer un segment de mémoire partagée (*shared memory*).

- Un **segment de mémoire partagée** est identifié de manière externe par un **nom** qui permet à tout processus possédant les droits, d'accéder à ce segment. Lors de la création ou de l'accès à un segment mémoire, un **numéro interne** est fourni par le système.
- Parmi les mécanismes de communication entre processus, l'utilisation de segments de mémoire partagée est la technique la plus rapide, car il n'y a pas de copie des données transmises. Ce procédé de communication est donc parfaitement adapté au partage de gros volumes de données entre processus distincts.

Frères des Écoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Mise en oeuvre d'un segment de mémoire partagée

- Si deux processus écrivent et lisent "en même temps" dans une même zone de mémoire partagée, il ne sera pas possible de savoir ce qui est réellement pris en compte. Le **segment de mémoire partagée** est considéré comme une **section critique**.
- D'où l'obligation d'utiliser des **sémaphores** pour ne donner l'accès à cette zone qu'à un processus à la fois.

API IPC System V

Les IPC **System V** sont une ancienne API (orientée UNIX).

	Files de messages	**Mémoire partagée**	**Sémaphores**
Headers	sys/types.h sys/ipc.h sys/msg.h	sys/types.h sys/ipc.h sys/shm.h	sys/types.h sys/ipc.h sys/sem.h
Création / Ouverture	msgget()	shmget()	semget()
Contrôle	msgctl()	shmctl()	semctl()
Opérations	msgsnd() msgrcv()	shmat() shmdt()	semop()
Structures associées	msqid_ds	shmid_ds	semid_ds

Deux opérations sont disponibles à partir du *shell* :

- `ipcs` : consultation des objets de communication.
- `ipcrm` : suppression des objets de communication.

Frères des Écoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

IPC POSIX

Les IPC **POSIX** sont une API plus récente et normalisée.

- les appels systèmes pour les files de messages sont : `mq_open()`, `mq_send()`, `mq_receive()`, `mq_close()` et `mq_unlink()`
- les appels systèmes pour les segments de mémoire partagé sont : `shm_open()`, `mmap()`, `munmap()`, `close()` et `shm_unlink()`
- les appels systèmes pour les sémaphores sont : `sem_open()`, `sem_post()`, `sem_wait()`, `sem_close()` et `sem_unlink()`
- Plus de détails : man mq_overview, man sem_overview, man shm_overview

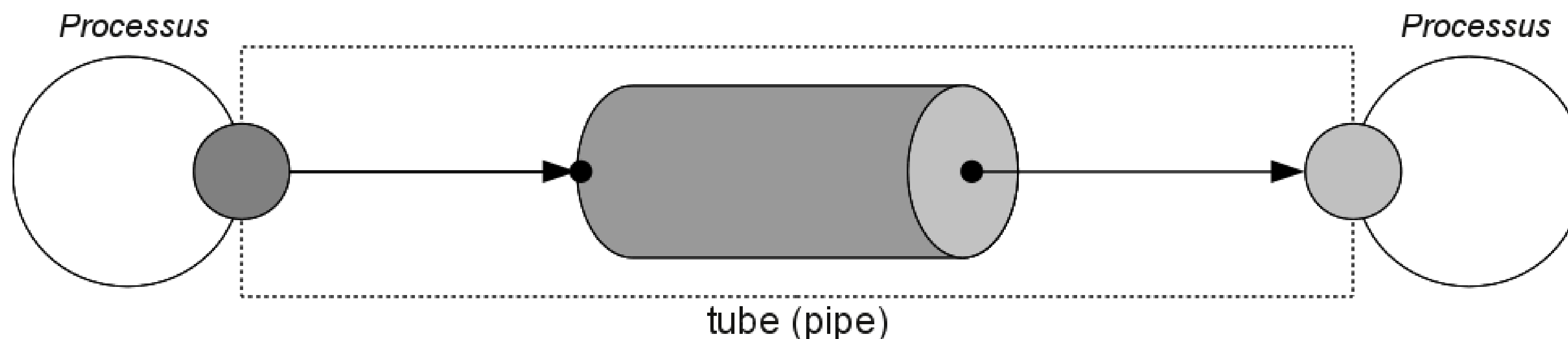
Les IPC POSIX fournissent une interface bien mieux conçue (portable et "*thread safe*") que celles de System V. D'un autre coté, les IPC POSIX sont moins largement disponibles (particulièrement sur d'anciens systèmes) que ceux de System V.

Frères des Écoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Les tubes

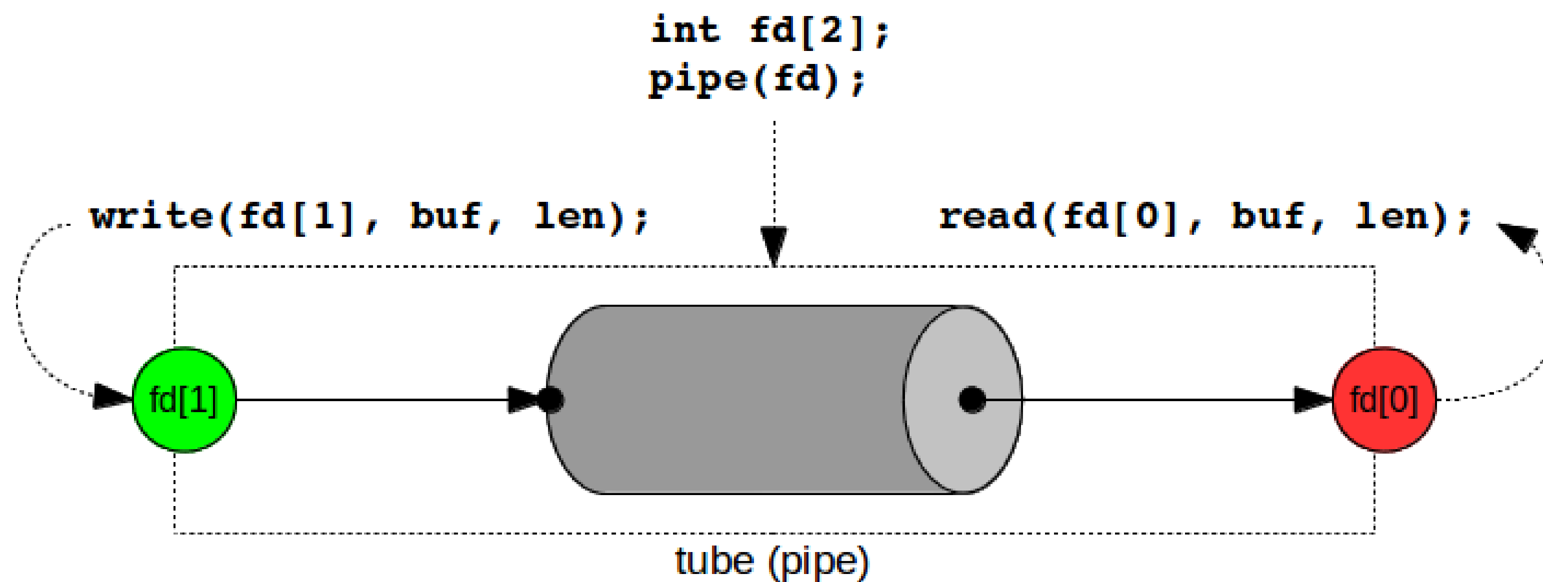
Les **tubes** (*pipe*) sont un mécanisme de communication entre **processus résidants sur une même machine**. On peut distinguer 2 catégories :

- les **tubes anonymes** (volatiles) : ils concernent des processus issus de la même application
- les **tubes nommés** (persistants) : ils concernent des processus totalement indépendants



Principe

Un **tube** peut être vu comme un **tuyau** dans lequel un processus, à une extrémité, produit des informations, tandis qu'à l'autre extrémité un autre processus les consomme.



Un tube est créé par l'appel système `pipe(int descripteur[2])` qui remplit le tableau `descripteur` avec les descripteurs de chacune des extrémités du tube (l'entrée en lecture seule et la sortie en écriture seule).

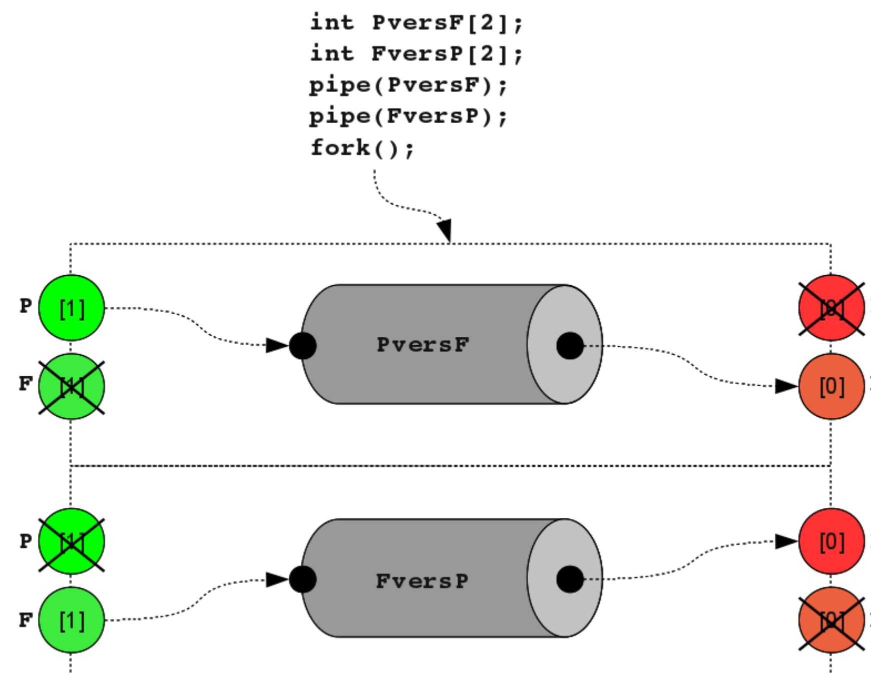
Caractéristiques

- intra-machine
- communication unidirectionnelle
- communication synchrone
- communication en mode flux (*stream*)
- remplissage et vidage en mode FIFO
- limité en taille

Frères des Écoles Chrétiennes
Saint-Jean-Baptiste de la Salle - Avignon

Communication bidirectionnelle

Les tubes étant des systèmes de communication unidirectionnels, ils faut créer **deux tubes** et les employer dans le sens opposé si l'on souhaite réaliser une communication bidirectionnelle entre 2 processus.



La création de ces deux tubes est sous la responsabilité du processus père. Ensuite, la primive `fork()` réalisant une copie des données dans le processus fils, il suffit pour le père et le fils de fermer les extrémités de tubes qu'ils n'utilisent pas.

Saint-Jean-Baptiste de la Salle - Avignon

Les tubes nommés

Un **tube nommé** est simplement un noeud dans le système de fichiers. Le concept de tube a été étendu pour disposer d'un **nom** dans ce dernier. Ce moyen de communication, disposant d'une représentation dans le système de fichier, peut être utilisé par des processus indépendants.

- La création d'un tube nommé se fait à l'aide de la fonction `mkfifo()`.
- La suppression d'un tube nommé s'effectue avec `unlink()`.
- Une fois le noeud créé, on peut l'ouvrir avec `open()` avec les restrictions dues au mode d'accès.
- Si le tube est ouvert en lecture, `open()` est bloquante tant qu'un autre processus ne l'a pas ouvert en écriture. Symétriquement, une ouverture en écriture est bloquante jusqu'à ce que le tube soit ouvert en lecture.
- Il est possible d'utiliser la fonction `fdopen()` qui retourne un flux associé au descripteur passé en paramètre. On peut ensuite utiliser les fonctions `fread()`, `fwrite()`, `fscanf()` ou `fprintf()` ...

Les signaux

Un **signal** peut être imaginé comme une sorte d'impulsion qui oblige le processus cible à prendre immédiatement une mesure spécifique.

Le rôle des signaux est de permettre aux processus de communiquer. Ils peuvent par exemple :

- réveiller un processus
- arrêter un processus
- avertir un processus d'un événement

L'utilisation des signaux sous Linux est décrite dans : `man 7 signal`

Frères des Écoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Envoi et réception d'un signal

- La commande `kill` permet d'envoyer un signal à un processus ou à un groupe de processus
- L'appel système `kill()` peut être utilisé pour envoyer n'importe quel signal à n'importe quel processus ou groupe de processus

À la réception d'un signal, le processus peut :

- l'ignorer
- déclencher l'action prévue par défaut sur le système (souvent la mort du processus)
- déclencher un traitement spécial appelé *handler* (gestionnaire du signal)

Frères des Écoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Interception d'un signal

- La primitive `signal()` permet d'associer un traitement à la réception d'un signal d'interruption. Une autre primitive, `sigaction()`, permet de faire la même chose et présente l'avantage de définir précisément le comportement désiré et de ne pas poser de problème de compatibilité.
- `signal()` installe le gestionnaire *handler* pour le signal *signum*. *handler* peut être `SIG_IGN` (ignoré), `SIG_DFL` (défaut) ou l'**adresse d'une fonction** définie par le programmeur (un « gestionnaire de signal »).

L'attente d'un signal démontre tout l'intérêt du multi-tâche. En effet, au lieu de consommer des ressources CPU inutilement en testant la présence d'un événement dans une boucle `while`, on place le processus en sommeil (*sleep*) et le processeur est mis alors à la disposition d'autres tâches.

Frères des Écoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Quelques signaux

- La commande `kill -l` liste l'ensemble des signaux disponibles
- Le fichier *header* `signum.h` déclare la liste des signaux :
  `find /usr -name signum.h` puis
  `grep SIG /usr/include/x86_64-linux-gnu/bits/signum.h`

```
#define SIGINT 2 /* Interrupt = Ctrl-C (ANSI) */...
#define SIGKILL 9 /* Kill, unblockable (POSIX) */...
#define SIGUSR1 10 /* User-defined signal 1 POSIX */...
#define SIGUSR2 12 /* User-defined signal 2 POSIX */...
#define _NSIG 65 /* Biggest signal number + 1 */
```

IPC

L'API Win32 de Microsoft propose différents moyens pour faire **communiquer plusieurs processus ou *threads***.

Les mécanismes fournis sont décrits dans la MSDN dans l'article « *Interprocess Communications* » : http ://msdn.microsoft.com/en-us/library/aa365574(v=VS.85).aspx.

Cela couvre l'utilisation du presse-papier (*clipboard*) jusqu'au *socket*.

L'API Win32 de Microsoft propose différents moyens pour **coordonner l'exécution de plusieurs *threads***. Les mécanismes fournis sont :

- les exclusions mutuelles
- les sections critiques
- les sémaphores

Frères des Écoles Chrétiennes
Saint-Jean-Baptiste de la Salle - Avignon

Fichier mappé

La technique officielle préconisée par Microsoft pour **partager de la mémoire** est d'utiliser les différentes fonctions pour accéder à un **fichier mappé (*FileMapping*)**.

- Un des processus ou *threads* doit créer en espace de mémoire partagée avec un fichier mappé en utilisant `CreateFileMapping()`.
- Les autres processus ou *threads* peuvent accéder à la mémoire partagée en l'ouvrant avec la fonction `OpenFileMapping()`.
- Pour avoir un pointeur utilisable pour accéder à une zone de la mémoire, il faut utiliser `MapViewOfFile()`. Après utilisation, il faut appeler `UnmapViewOfFile()` pour détacher le *handle*.
- Pour libérer la mémoire, il suffit d'appeler `CloseHandle()` sur le *handle* obtenu.

L'accès simultanée à la mémoire partagée doit être sécurisé par l'utilisation d'un *mutex* par exemple.

Mutex

Le **mutex** est de type **HANDLE**, comme tout objet ressource dans l'API Win32. Les fonctions utilisables sont :

- Création du *mutex* : `CreateMutex()`
- Opération V(), libération ou déverrouillage du *mutex* : `ReleaseMutex()`
- Opération P(), prise ou verrouillage du *mutex* : `WaitForSingleObject()`

L'objet **section critique** fournit une synchronisation similaire au *mutex* sauf que ces objets doivent être utilisées par les *threads* d'un même processus. Ce mécanisme est plus performant que les *mutex* et il est plus simple à utiliser.

Frères des Écoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Section critique

Typiquement, il suffit de déclarer une variable de type `CRITICAL_SECTION`, de l'initialiser grâce à la fonction `InitializeCriticalSection()` ou `InitializeCriticalSectionAndSpinCount()`. Les fonctions utilisables sont :

- Le *thread* utilise la fonction `EnterCriticalSection()` ou `TryEnterCriticalSection()` avant d'entrer dans la section critique.
- Le *thread* utilise la fonction `LeaveCriticalSection()` pour rendre le processeur à la fin de la section critique.
- Chaque *thread* du processus peut utiliser la fonction `DeleteCriticalSection()` pour libérer les ressources système qui ont été allouées quand la section critique a été initialisé. Après son appel, aucune fonction de synchronisation ne peut plus être appelée.

Sémaphore

Sous Windows, un **sémaphore** est un **compteur qui peut prendre des valeurs entre 0 et une valeur maximale** définie lors de la création du sémaphore. Le sémaphore est de type **HANDLE**. Les fonctions utilisables sont :

- La fonction `CreateSemaphore()` crée l'objet sémaphore.
- La fonction `OpenSemaphore()` ouvre un objet sémaphore déjà créé par un autre thread ou processus.
- Opération V() : la fonction `ReleaseSemaphore()` libère un jeton.
- Opération P() : la fonction `WaitForSingleObject()` permet de prendre un sémaphore.

Frères des Écoles Chrétiennes
Saint-Jean-Baptiste de la Salle - Avignon