# Unix : concepts et outils avancés
# TD no 10 : mémoire partagée, synchronisation, signaux

Olivier Aumage
LIP, bureau 302
email : `Olivier.Aumage@ens-lyon.fr`

6 décembre 2001

## 1 Préambule

Finir le TD numéro 9 s'il n'est pas terminé.

## 2 Signaux

Cet exercice a pour but de réaliser une communication utilisant le code *morse* (ou un quelconque code similaire) entre deux processus — un client et un serveur — au moyen de signaux. Un résumé du codage morse est disponible dans le fichier `morse.txt` du répertoire `/home/oaumage/Unix`. Pour information, il faut y ajouter les particularités suivantes :

– les signes morses sont séparés par un blanc ;
– les lettres sont séparées par trois blancs ;
– les mots sont séparés par sept blancs.

**Questions**

– Écrivez un programme `tstsig_1.c` qui boucle infiniment et affiche :
  – `'.'` lorsqu'il reçoit un signal `SIGUSR1`;
  – `'-'` lorsqu'il reçoit un signal `SIGUSR2`;
  – `' '` lorsqu'il reçoit un signal `SIGALRM`.

  Vous utiliserez la fonction `sigaction(3)` pour gérer la mise en place des gestionnaires (*handler*) de signaux. Vous pourrez utiliser la commande `kill(1)` pour envoyer des signaux au processus depuis un terminal.
– Écrivez un programme `tstsig_2.c` à partir de `tstsig_1.c` qui — en plus des fonctionnalités de `tstsig_1.c` — se termine en affichant le message `"[over]"` lorsqu'il reçoit un signal `SIGTERM`.
– Écrivez un programme `serveur.c` et un programme `client.c` avec les fonctionnalités suivantes :

  **serveur** Le serveur itère en demandant un message (directement en morse, ou en format texte si vous avez le temps de programmer la conversion en fin de TD) et un numéro de `pid`, puis envoie le message au processus client correspondant en utilisant les 4 signaux ci-dessus avec les mêmes conventions de signification.

  **client** Le client est une version étendue de `tstsig_2.c`. Il prend le `pid` du serveur en argument de ligne de commande puis reçoit et affiche un message envoyé par le serveur.

  Note : il est indispensable que le message soit transmis de manière fiable.

## 3 Mémoire partagée, fichiers mappés, synchronisation

Cette section prolonge les exercices du TD numéro 9 sur la mémoire partagée, les fichiers mappés et les commandes `mmap(2)` et `munmap(2)`.

L'objectif de cet exercice est de traiter la question classique de la synchronisation d'un producteur et d'un consommateur. On va utiliser deux programmes `prod.c` et `cons.c` qui communiquent via un segment de mémoire partagée (i.e. un fichier mappé) : le producteur écrit des messages dans la zone de mémoire partagée le plus vite possible, et `cons.c` lit les messages le plus vite possible. Chaque message a une longueur totale de 8 caractères et ne possède pas d'entête. La zone de mémoire partagée a une capacité de 16 messages plus 4 éléments de type `int` pour la synchronisation.

Le fichier `test_and_set.c` du répertoire `/home/oaumage/Unix` contient les primitives nécessaires à la réalisation de verrous fiables. La primitive `test_and_set` fonctionne sur le principe suivant :

- une variable `lock` de type `volatile int` est utilisée comme verrou (le terme `volatile` indique que la variable peut être modifiée de manière asynchrone), elle aura comme valeur 0 (libre) ou 1 (verrouillée) ;
- une instruction assembleur spéciale est utilisée par `test_and_set` pour *à la fois* mettre le verrou à la valeur 1 et retourner la valeur précédente ; le terme `à la fois` indique que ces deux opérations sont faites de manière atomique ;
- si la valeur *précédente* du verrou était déjà 1, cela signifie qu'un processus a déjà verrouillé le verrou, et il faut donc recommencer l'opération plus tard ;
- si la valeur *précédente* du verrou était 0, le verrou était libre (il est maintenant verrouillé), et la ressource correspondante peut être utilisée par le processus qui vient d'*acquérir* le contrôle du verrou ;
- la libération du verrou consiste simplement à remettre la valeur du verrou à 0.

### Questions

- Écrivez le programme `prod.c` et le programme `cons.c`. L'un des deux programmes devra en outre réaliser l'initialisation du fichier d'échange puis indiquer que l'autre programme peut être lancé.

— * —