

# Systemes d'Exploitation

## TP1 : *OS et Programmation*

Jean-Christophe Deneuille  
<jean-christophe.deneuille@xlim.fr>

17 mars 2016

L'objectif de ce TP est de mettre en évidence les mécanismes qui régissent le système d'exploitation à travers des exemples simples d'implémentation, en observant les différences de comportements de nos processus en fonction des modifications que l'on y apportera.

### Exercice 1 Localité Spatiale

Nous avons vu ce principe en cours : “lorsque une donnée a été utilisée à un instant  $t$ , il y a de fortes chance pour que les données qui lui sont proches (en mémoire) soient accédées à l'instant  $t + 1$ ”.

Nous allons mettre ce principe en évidence à l'aide d'un programme simple.

```
#include <stdio.h>

#define DIMENSION /* faire varier */

int tab[DIMENSION] [DIMENSION];

int main () {
    register unsigned int i, j;

    for(i=0 ; i<DIMENSION ; ++i) /*outer loop*/
        for(j=0 ; j<DIMENSION ; ++j) /*inner loop*/
            tab[i][j] = 0;

    return 0;
}
```

1. La bibliothèque `time.h` permet de mesurer (plus ou moins précisément) le temps qui s'écoule entre 2 instructions. Rajoutez au morceau de code ci-dessus de quoi estimer le temps qui s'écoule pour initialiser ce double tableau.

2. Dans votre programme modifié, j'appelle  $\mathcal{A}$  la configuration avec la boucle  $i$  à l'extérieur. Déterminez les temps d'exécution de  $\mathcal{A}$  en fonction de  $\text{DIMENSION}$  dans le Tableau ci-dessous.

DIMENSION	64	128	256	512	1024	2048	4096
temps( $\mathcal{A}$ )							
temps( $\mathcal{B}$ )							

3. J'appelle  $\mathcal{B}$  le programme obtenu en inversant les boucles interne (`inner`) et externe (`outer`). Remplissez de même le Tableau ci-dessus pour ce nouvel algorithme.
4. Que constatez-vous ? Selon vous, comment cette différence s'explique ?
5. Comment évolue cette différence de temps d'exécution en fonction de la dimension ?

## Exercice 2 Architecture aussi

Nous allons maintenant illustrer un autre phénomène impliquant le système d'exploitation — et l'architecture sous-jacente — impactant les performances d'un programme.

```
#include <stdio.h>

#define DIMENSION /* faire varier */

int tab[DIMENSION];

int main () {
    register unsigned int i;

    for(i=0 ; i<DIMENSION ; ++i)
        tab[i] += tab[i+1];

    return 0;
}
```

1. L'addition effectuée à la ligne `tab[i] += tab[i+1]` devrait se faire en un temps négligeable. Modifiez le programme pour afficher les temps de chaque exécution de cette ligne. Que constatez-vous ?
2. Modifiez votre programme de sorte à ce qu'il affiche le temps maximal pris pour l'exécution de cette ligne à la fin de votre programme. Ce temps est-il négligeable ?
3. Comment expliquez-vous ce résultat ? Notez que vous pouvez obtenir le code assembleur correspondant à votre programme C à l'aide de la commande `gcc -S mon_prog.c` (éventuellement avec des options d'optimisation).

### Exercice 3 man — Fais tourner

Une des différences majeures entre la *pagination* et la *segmentation* est que cette dernière permet d’avoir un accès plus fin à la gestion de la mémoire, du point de vue du programmeur. En particulier, il est possible de partager un même segment mémoire entre différents processus (nous reviendrons sur la communication inter-processus ultérieurement). Ceci constitue un moyen “simple” et *rapide* d’échanger des données. On parle alors de **shared memory**.

1. À quoi sert `shmget()` ?
2. À quoi sert `shmctl()` ?
3. À quoi sert `shmat()` ?
4. À quoi sert `shmdt()` ?
5. Construire une fonction `allouerSegmentPartage` prenant en entrée :
  - La taille du segment mémoire à partager,
  - Un nom de fichier associé au main appelant cette fonction,
  - Ainsi qu’une clé (entier) permettant d’identifier le segment.Votre fonction affichera l’identificateur et la clé du segment partagé créé, et retournera cet identificateur.

Vous aurez besoin des bibliothèques système `types.h` (pour les clés notamment), `ipc.h` (pour la communication inter-processus), et `shm.h` pour la mémoire partagée.