

# Systemes d'Exploitation

## TP3 : *Ordonnancement*

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

31 mars 2016

Dans ce TP, nous allons reprendre le code de base du TP précédent (voir page suivante), et rajouter une couche d'ordonnancement afin de s'assurer que nos processus s'exécutent bien dans l'ordre voulu.

Dans cet exercice, vous aurez besoin des bibliothèques du TP précédent, plus les bibliothèques `fcntl.h` (encore des constantes) et `semaphore.h` (pensez à linker vers la librairie temps réel (`-lrt`)<sup>1</sup>).

L'instruction `ipcs` vous permettra d'obtenir des informations sur la mémoire partagée fraîchement créée.

### Exercice 1 I am your father

Reprenez le code que vous avez utilisé pour le **TP2**. Outre les fonctions de création et de destruction de segments mémoire partagés, nous avons un programme principal `main` qui mutait en un processus père et un processus fils (`fork()`). Le processus fils écrivait dans le segment partagé, et le père lisait le contenu de ce segment. Des exécutions successives de ce programme montraient bien que nous ne contrôlions pas l'ordonnancement de ces 2 processus (et donc que le père pouvait lire un contenu de mémoire partagée vide, le fils n'ayant alors pas encore écrit dedans).

1. Afin de régler ce problème, nous allons utiliser un sémaphore. Commencez par consulter les **manpages**<sup>2</sup> de `sem_overview` ainsi que celles des fonctions associées `sem_open`, `sem_post`, `sem_wait` et `sem_close`.
2. Maintenant que vous êtes familier avec la notion de sémaphore, créez-en un avant le `fork()` dans votre `main` (rajoutez immédiatement la partie de code à la fin du `main` pour le fermer, sinon ... ?).
3. Que devez-vous rajouter dans la partie du code concernant le processus fils ? À quel endroit ?
4. Mêmes questions pour le processus père ?

---

1. ou utilisez `-pthread` si la librairie temps réel n'est pas disponible...

2. Ici par exemple pour ceux allergiques à l'anglais...

```

1  int main()
2  {
3      int pid;
4      char *mem;
5      int shmid;
6      int flag=0;
7      char *name = "TP2.cpp";
8      /*Creer le segment partage*/
9      shmid = allouerSegmentPartage(100,name,1);
10     switch(pid=fork()) {
11         case -1:
12             fprintf(stderr,"Where are you, Luke ?");
13             return -1;
14         case 0:
15             printf("My name is Luke %d\n",getpid());
16             mem = shmat(shmid,0,flag);
17             if(mem==(char*)-1) {
18                 fprintf("Attachement impossible");
19                 exit(1);
20             }
21             strcpy(mem,"Anakin, I am your son.");
22             if(shmdt(mem)==-1) {
23                 fprintf(stderr,"Detachement impossible");
24                 exit(1);
25             }
26             return 0;
27         default:
28             printf("I am the father.");
29             mem = shmat(shmid,0,flag);
30             if(mem ==(char*)-1) {
31                 fprintf(stderr, "Attachement impossible");
32                 exit(1);
33             }
34             printf("Luke a ecrit :\n%s\n",mem);
35             if(shmdt(mem)==-1) {
36                 fprintf(stderr, "Detachement impossible");
37                 exit(1);
38             }
39     }
40     /*Le pere attend la mort du processus fils*/
41     wait(0);
42     /*Detruire le segment partage*/
43     detruireSegmentPartage(shmid) ;
44     return 0;
45 }

```