# TP 7(3h): Synchronisation

*Remarque : Utiliser les chapitres 4 & de Advanced Linux Programming comme support, ainsi que les pages de manuel de****semget,pthread\_create, pthread\_join, pthread\_mutex\_lock, pthread\_barrier\_wait****, etc.*

**Point de départ :** *Pour répondre aux questions suivantes vous aurez besoin des sources du TP6.*

1. **Synchronisation de processus de par sémaphore**

*Dans cet exercice, nous partirons de la solution à l’Ex.1 du TP6.*   
Dans le TP 6, les 15s de **sleep** de P1 servent à permettre à l’utilisateur le lancement de P2 pendant cet intervalle. Ceci n’est pas une bonne solution pour la synchronisation. On voudrait que P2 attende que les données de la matrice soient prêtes avant de les utiliser. Ceci peut se faire en utilisant un sémaphore pour synchroniser P1 et P2.

A l’aide de la librairie *semops.[hc]* fournie en annexe, modifier les programmes P1 et P2 pour qu’ils allouent un sémaphore (avec **semget**), l’initialisent et l’utilisent correctement pour la synchronisation. P2 pourra libérer le sémaphore à la fin du programme.

*Attention : vous devez être en mesure d’expliquer le code des fonctions de semops.c.*

*A noter : les sémaphores peuvent servir aussi à synchroniser des threads.*

1. **Synchronisation de threads par mutex**

*Dans cet exercice, nous partirons de la solution à l’Ex.2 du TP6.*   
Nous souhaitons maintenant calculer la puissance 4 de la matrice, en rajoutant un thread (3) :

* Tout comme le thread 2, le thread 3 calcule le carré de la matrice.
* A la fin du calcul, chacun des deux threads (2 et 3) replace le résultat dans la variable A, ainsi le second thread utilisera le résultat produit par le premier.
* Il ne faut cependant pas que les deux calculs aient lieu en même temps. Pour éviter cela, rajouter une synchronisation par mutex (fonctions ***pthread\_mutex\_lock*** et ***pthread\_mutex\_unlock***).

1. **Synchronisation par barrière**

Nous souhaitons à présent paralléliser le calcul du produit. Ecrire une fonction parallel\_square\_a\_square\_matrix qui reprend le code de square\_a\_square\_matrix mais utilise S2 threads(pour une matrice de taille S), chacun d’entre eux calculant un élément M\_ELEMENT(\*result,row,col)du résultat. La boucle for intérieure sera donc exécutée en parallèle par les S2 threads.

Remarques :

1. Les threads crées par parallel\_square\_a\_square\_matrix devront prendre en entrée plusieurs paramètres : les pointeurs des matrices source/résultat, les valeurs de *row* et *col*. Etant donné qu’un thread ne prend qu’un seul paramètre, de type void\*, il faudra créer une structure pour stocker ces paramètres et passer en le pointeur vers cette structure.
2. La fonction parallel\_square\_a\_square\_matrix ne doit se terminer qu’à la fin de ces calculs. Pour cela, mettre en place une synchronisation par barrière (fonction ***pthread\_barrier\_wait***et fonctions liées).

Tester la nouvelle version de parallel\_square\_a\_square\_matrix avec le programme de l’exercice 2.