TP 4 : Synchronisation & Exclusion Mutuelle

Sémaphores & Threads

Sous UNIX

Sémaphores

Les sémaphores représentent l'outil le plus utilisé pour la résolution des problèmes de synchronisation (voir Chapitre 2).

Syntaxe

Header : `semaphore.h // bibliothèque des sémaphores`

```
sem_t  S; // declaration des sémaphores

sem_init(&S , 0, valeur_initiale); // initialisation des sémaphores

sem_wait(&S);  // primitive p(S)

sem_post(&S); // primitive v(S)
```

Threads

Les threads ou les processus de poids faible sont des unités élémentaires d'exécution. Le multi-threading permet d'exécuter les threads comme des processus de poids lourds (voir Chapitre 1).

Syntaxe

Header : `pthread.h // bibliothèque des threads`

```
pthread_t  Th; // délaratioon d'un thread

pthread_create(&Th, NULL,procedure_thread,NULL); // création d'un
thread

pthread_join(Th, NULL); // lancement d'un thread

gcc code_source.c -pthread -o executable  //  compilation avec des
threads
```

Problème du Producteur/Consommateur :

Reprenons le problème du Producteur/Consommateur vu au cours. Ecrire le code du producteur et du consommateur qui s'exécuteront sous forme de deux threads et se synchroniseront au moyen des sémaphores. Les algorithmes des threads producteur et consommateur ainsi que les variables globales du problème sont définis dans le support de cours (Chap2 – Sémaphores appliqués à la synchronisation – Exemple du Producteur/Consommateur). La mémoire tampon peut être considérée comme un tableau d'entiers. Les objets peuvent être considérés comme de simples entiers.

Exécutez le processus plusieurs fois. Que constatez-vous ?