Faculté des Sciences & Techniques — Université de Limoges

Licence 3ème année

Programmation Concurrente

TD n°2

Processus de poids léger ou « threads *» & Sémaphore*

Processus de poids léger

Les threads POSIX s'utilisent au travers d'un certain nombre d'instructions :

```
1 pthread_t         a_thread;
2 pthread_attr_t     a_thread_attribute;
3
4 void thread_function(void *argument);
5 char *some_argument;
6 pthread_create(&a_thread, a_thread_attribute,
7              (void*)&thread_function, (void*)&some_argument);
```

1 – Soit le programme suivant, commentez son exécution :

```
1  void print_message_function( void *ptr );
2  main()
3  {
4        pthread_t thread1, thread2;
5        char *message1 = "Hello";
6        char *message2 = "World";
7
8        pthread_create(&thread1, NULL, (void *)print_message_function, (void *)message1);
9        pthread_create(&thread2, NULL, (void *)print_message_function, (void *)message2);
10       exit(0);
11 }
12 void print_message_function( void *ptr )
13 {
14        char *message;
15        message = (char *) ptr;
16        printf("%s ", message);
17 }
```

Est-ce que l'affichage va se produire au final ?

2 – Voici une version améliorée du programme précédent, quelles sont ces améliorations ?

```
1  void print_message_function( void *ptr );
2  main()
3  {
4     pthread_t thread1, thread2;
5     char *message1 = "Hello";
6     char *message2 = "World";
7     pthread_create(&thread1, NULL, (void *) print_message_function, (void *) message1);
8     sleep(10);
9     pthread_create(&thread2, NULL, (void *) print_message_function, (void *) message2);
10    sleep(10);
11    exit(0);
12 }
13 void print_message_function( void *ptr )
14 {
15    char *message;
16    message = (char *) ptr;
17    printf("%s", message);
18    pthread_exit(0);
19 }
```

Sémaphore

Pour l'utilisation des sémaphores en programmation C :

```
 1 #include <semaphore.h>
 2
 3 int main()
 4 {
 5    sem_t semaphore;
 6    int compteur = 0;
 7    sem_init( &semaphore, 0, 1 ); /* On crée la sémaphore avec la valeur 1*/
 8    sem_wait( &semaphore ); /* On « prend » la sémaphore */
 9    compteur++;
10    sem_post( &semaphore ); /* On « libère » la sémaphore */
11    sem_destroy( &semaphore );
12 }
```

Il faut également compiler en reliant la bibliothèque pthread : `-lpthread`.

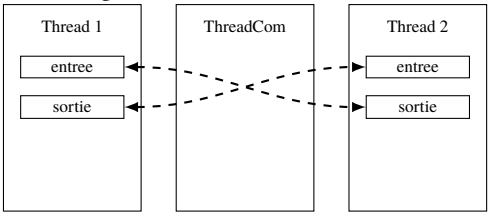
3 – Dans un programme, on utilise des threads numérotées de 1 à 5, en plus du programme principal.
On voudrait, dans le programme principal, déclencher un traitement uniquement lorsque le travail fait dans chaque thread prend fin (après que les threads 1 à 5 aient terminé).

Indiquez comment, à l'aide de sémaphores, il est possible de le faire.

4 – Dans un programme, on utilise deux threads T_1 et T_2.
On voudrait créer un nouveau programme où le fonctionnement de ces deux threads est alterné : T_1 puis T_2 puis T_1 puis T_2 *etc*.

Indiquez ce qu'il faudrait ajouter aux threads 1 et 2 pour arriver à ce fonctionnement.

5 – On veut permettre à deux threads de **communiquer par l'intermédiaire** d'une troisième.
Soient « Thread1 » et « Thread2 », les threads de traitement et « ThreadCom » la thread chargée d'effectuer les échanges de données.



Chacune des threads, « Thread1 » et « Thread2 », possèdent deux variables :

- ▷ *entree* : pour la consultation des données en provenance de l'autre thread ;
- ▷ *sortie* : pour la mise à disposition de données à destination de l'autre thread.

Le fonctionnement de « ThreadCom » est le suivant, lorsqu'elle s'exécute :

i. elle échange les contenues des variables d'entrée et de sortie entre « Thread1 » et « Thread2 » ;
ii. elle recommence ;

Indiquez comment à l'aide de sémaphores, il est possible d'organiser les communications :

- ⋆ en protégeant l'accès concurrent aux variables *entrée* et *sortie* de chaque thread ;
- ⋆ en synchronisant les opérations de lecture et écriture.

Justifiez votre solution par l'utilisation de schémas d'interactions temporelles (*Message Sequence Chart*).