TP9

Page web du cours :

`http://www.lsv.ens-cachan.fr/~schwoon/enseignement/systemes/ws1516/`.

Pensez à utiliser `man` pour découvrir les détails des appels système et des commandes dans le shell.

1 Sémaphores

Rappelez-vous que les threads d'un même processus partagent leur mémoire (sauf la pile) ; notamment ils partagent les variables statiques et dynamiquement allouées. On considère le programme suivant (`counter.c`):

```
#include <pthread.h>
#include <stdio.h>

#define MAX 100000
int ctr = 0;

void* count(void* data) {
  int i;
  for (i = 0; i < MAX; i++) ctr++;
}

int main () {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, count, NULL);  // create first thread
  pthread_create(&t2, NULL, count, NULL);  // create second thread
  pthread_join(t1, NULL);  // wait for first thread
  pthread_join(t2, NULL);  // wait for second thread
  printf("Compteur: %d\n", ctr);
}
```

(a) Quel est le résultat attendu pour ce programme ?

(b) Qu'est-ce qui peut empêcher ce résultat attendu ?

(c) Quels résultats sont (théoriquement) possibles ?

Les *sémaphores* représentent une solution pour résoudre le problème des accès concurrents aux variables. Un sémaphore est une strucutre de données utilisée pour la synchronisation et pour assurer un accès coordonnée aux ressources partagés. En C, les sémaphores sont définis par le fichier `semaphore.h`. Les fonctions les plus importantes pour nos objectifs seront les suivantes :

- `int sem_init(sem_t *sem, int pshared, unsigned int value);`
  Initialise `sem` avec la valeur donnée ce qui donne le nombre de créneaux disponible pour accéder à une ressource au même temps. Pour nos objectifs, `pshared` devrait être `NULL`.
- `int sem_wait(sem_t *sem);`
  Si le sémaphore a valeur 0 cette fonction attend qu'on créneau devient disponible. Sinon, elle décroît sa valeur immédiatement.
- `int sem_post(sem_t *sem);`
  Augmente la valeur de la sémaphore (un créneau devient disponible).

Étudiez les pages `man` pour en savoir plus.

(d) Utilisez les sémaphores pour réparer le programme `counter.c`.

Enfin, considérons le programme suivant (`order.c`) :

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>

void* first(void* data)  { printf("First\n");  }
void* second(void* data) { printf("Second\n"); }
void* third(void* data)  { printf("Third\n");  }

int main () {
  pthread_t t1, t2, t3;

  pthread_create(&t3, NULL, third, NULL);
  pthread_create(&t2, NULL, second, NULL);
  pthread_create(&t1, NULL, first, NULL);

  // wait for all threads
  pthread_join(t1, NULL);  pthread_join(t2, NULL); pthread_join(t3, NULL);
}
```

Modifiez le programme pour assurer que le programme donne toujours la sortie suivant :

```
First
Second
Third
```

(e) Écrivez un programme en C avec deux threads. Le premier thread affiche les entiers pairs jusquà 100, le deuxième les impairs. Utilisez des sémaphores pour assurer le bon ordre des entiers.

2 Producer/consumer

Un problème typique dans la programmation concurrente est celui du *producteur et consommateur* : un processus produit des données que l'autre consomme. On considère le programme `procon.c`. Ici, le producteur obtient des caractères dans un fichier et les envoie vers un espace partagé avec le processus consommateur qui les affiche sur l'écran. Cependant, dans la version actuelle, aucune synchronisation existe entre les deux.

Modifiez le programme tel qu'il affiche le contenu d'un fichier correctement.

3 Mandelbrot

Soit c un nombre complexe. On considère la série $z_0 := 0$ et $z_{n+1} := z_n^2 + c$ pour $n \geq 0$. L'*ensemble de Mandelbrot* est défini comme l'ensemble des valeurs c telles que la série des z_n est bornée. On sait que cela est le cas si z_n ne sort jamais d'un cercle de radius 2 autour de 0. Si jamais la série sort de ce cercle, soit $m(c)$ le plus petit indice n tel que c'est le cas.

Une application populaire pour $m(c)$ est de créer de jolies images ; on associe l'écran avec un rectangle et chaque pixel avec la valeur c qui y correspond ; le pixel est ensuite peint avec une couleur correspondant à $m(c)$.

La page web du cours contient un tel programme. Votre tâche consiste en l'accelerant en lançant plusieurs threads en parallèle. Les machines de la salle 411 sont équipées de plusieurs cœurs (utilisez `cat /proc/cpuinfo` pour trouver plus d'information). Chaque thread va donc travailler sur une partie différent de l'image.

Note: Compilez le programme avec `make`.

Note (2): Cette exercice ne nécéssite pas des sémaphores.

Pages man utiles : pthreads(7), pthread_create(3), pthread_join(3).