Informatique concurrente et répartie

Les processus Légers POSIX

Emmanuel Romagnoli

Plan

- La gestion des threads
- Les sémaphores

La gestion des threads

La fonction pthread_create

Pour créer un nouveau processus léger au sein d'un processus lourd, on utilise la fonction `pthread_create` qui est définie dans `pthread.h`.

La signature de cette fonction est la suivante :

```
int pthread_create (
                    pthread_t       *thread,
                    const pthread_attr_t *attr,
                    void *(*start_routine)(void*),
                    void *arg
                   )
```

La fonction pthread_create

Le premier paramètre est un pointeur sur une structure `pthread_t` (autrement dit un tableau d'éléments de type `pthread_t`).

Cette structure (définie dans pthread.h) permet de gérer le ou les threads.

Le deuxième paramètre est un pointeur sur une structure de type `pthread_attr_t` dans laquelle sera enregistrés les attributs de fonctionnement des threads. Si ce pointeur est mis à `NULL`, les attributs par défaut sont utilisés

La fonction pthread_create

Le troisième paramètre est un pointeur sur la fonction qui doit être exécutée au sein du ou des threads.

Cette fonction doit obligatoirement prendre en paramètre un pointeur sur void et ne rien renvoyer. Sa signature sera donc :

```
void nomFonction (void* pt)
```

Le dernier paramètre est un pointeur sur `void` qui permet de désigner une zone de mémoire qui sera utilisée par la fonction (grâce à sa variable locale `pt`). Cette zone permettra de passer des paramètres à la fonction et/ou de récupérer des résultats.

La fonction pthread_create

Cette fonction renvoie un entier qui peut prendre deux valeurs :

- 0 en cas de succès ;
- une valeur supérieure à 0 en cas d'échec.

Exemple 1

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

void *fct (void *pt)
 {
  printf ("fct est executee par le thread\n");
 }

int main (int c, char *v[])
 {
  pthread_t thread;

  pthread_create (&thread, NULL, fct, NULL);

  sleep (1);

  return 0;
 }
```

Exemple 1

On compile le programme en ajoutant l'option « -lpthread » afin de faire le lien entre notre programme et la bibliothèque pthread.

```
AIL/Cours/2012/3IC - Informatique concurrente
12/3IC - Informatique concurrente$ gcc exemple1ThreadPOSIX.c -lpthread -o exemple1ThreadPOSIX
12/3IC - Informatique concurrente$ ./exemple1ThreadPOSIX

12/3IC - Informatique concurrente$
```

Remarque : sur des distributions telles que Ubuntu, il est peut être nécessaire de faire au préalable

« `sudo apt-get install build-essential manpages-dev linux-headers-generic dkms` »

Exemple 2

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

void *fct (void *pt)  {
  int i;
  for (i=0; i< *((int*) pt); i++) *( ((int*) pt)+1) += i;
  printf ("Le thread a calcule la somme de %d entiers\n",
       *((int*) pt) );     }

int main (int c, char *v[])
 {
  pthread_t thread;
  int       val [] = {10, 0};
  pthread_create (&thread, NULL, fct, val);
  sleep (1);
  printf ("Le thread a determine que la somme est egale a
           %d\n", val[1] );
  return 0;  }
```

Exemple 1

On compile ce second programme en utilisant encore l'option « -lpthread » et on constate que le pointeur pt correspond au quatrième argument de l'instruction pthread_create.

On peut donc utiliser le tableau pour passer des paramètres et récupérer des résultats.

```
emmanuel@emmanuel-VirtualBox:/media/sf_C_DRIVE/TRAVAIL/Cours/2012/3IC - Informatique concurrente$ gcc exemple2ThreadPOSIX.c -lpthread -o exemple2ThreadPOSIX
emmanuel@emmanuel-VirtualBox:/media/sf_C_DRIVE/TRAVAIL/Cours/2012/3IC - Informatique concurrente$ ./exemple2ThreadPOSIX
Le thread a calcule la somme de 10 entiers
Le thread a determine que la somme est egale a 45
emmanuel@emmanuel-VirtualBox:/media/sf_C_DRIVE/TRAVAIL/Cours/2012/3IC - Informatique concurrente$
```

La fonction pthread_join

Jusqu'à présent, on a endormi le thread père (le main) afin de laisser le temps au thread fils pour s'exécuter.

Cette solution n'est pas fiable car elle implique de faire une hypothèse, sur le temps d'exécution du thread, qui peut se révéler fausse.

La bonne solution consiste donc à mettre en place une barrière de synchronisation (à l'instar de la fonction `wait` pour les processus lourds)

La fonction pthread_join

La signature de cette fonction est la suivante :

```
int pthread_join (pthread_t thread, void** retval)
```

Le premier paramètre est la structure identifiant le thread sur lequel on doit se placer en attente.

Le second paramètre est un pointeur permettant de stocker une information retournée par le thread grâce à la fonction `pthread_exit`. Ce pointeur peut être mis à `NULL`.

La fonction pthread_exit

Cette fonction permet d'arrêter correctement le thread et d'envoyer une information vers le thread père.

La signature de cette fonction est la suivante :

```
int pthread_exit (void* retval)
```

Le paramètre est un pointeur désignant une zone de mémoire (variable, structure, tableau...) contenant des informations à destination du processus père.

Exemple

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

void *fct (void *pt)
 {
  int i;
  int* res = (int*) malloc (sizeof(int));
  for (i=0; i< *((int*) pt); i++) *res += i;
  printf ("Le thread a calcule la somme de %d entiers\n",
          *((int*) pt) );
  printf ("Le thread met le resultat a l'adresse %p\n", res );
  pthread_exit ((void*) res);
 }
```

Exemple

```
int main (int c, char *v[])
 {
  pthread_t thread;
  int       val = 10;
  int*      res;

  pthread_create (&thread, NULL, fct, &val);
  pthread_join (thread, (void**) &res);

  printf ("Le thread recupere le resultat a l'adresse %p\n",
             res );
  printf ("Le thread a determine que la somme est egale a
             %d\n", *res );

  if (res) free (res);
  return 0;
 }
```

La fonction pthread_exit

On compile et on exécute.

```
emmanuel@emmanuel-VirtualBox:/media/sf_C_DRIVE/TRAVAIL/Cours/2012/3IC - Informatique concurrente$ gcc exemple3ThreadPOSIX.c -lpthread -o exemple3ThreadPOSIX
emmanuel@emmanuel-VirtualBox:/media/sf_C_DRIVE/TRAVAIL/Cours/2012/3IC - Informatique concurrente$ ./exemple3ThreadPOSIX
Le thread a calcule la somme de 10 entiers
Le thread met le resultat a l'adresse 0x7f39d80008c0
Le thread recupere le resultat a l'adresse 0x7f39d80008c0
Le thread a determine que la somme est egale a 45
emmanuel@emmanuel-VirtualBox:/media/sf_C_DRIVE/TRAVAIL/Cours/2012/3IC - Informatique concurrente$
```

La fonction pthread_cancel

Cette fonction permet d'envoyer un signal vers un thread afin de lui demander son arrêt. Sa signature est :

```
int pthread_cancel (pthread thread)
```

Le thread cible peut interpréter ce signal d'arrêt de trois manière :

- interrompre immédiatement son fonctionnement ;
- retarder l'interruption jusqu'à ce qu'un point d'annulation soit atteint (dans la fonction associée au thread cible) ;
- ignorer le signal d'interruption

La fonction pthread_setcancelstate

On utilise `pthread_setcancelstate` au sein de la fonction associée au thread afin de modifier sa sensibilité au signal d'interruption. Sa signature est :

```
int pthread_setcancelstate (int s, int* os)
```

Le premier argument correspond au nouvel état :

- `PTHREAD_CANCEL_ENABLE` (le thread tient compte des demandes d'arrêt)
- `PTHREAD_CANCEL_DISABLE` (le thread ignore les demandes d'arrêt).

Le second argument permet de récupérer l'état précédent.

La fonction pthread_setcanceltype

On utilise `pthread_setcanceltype` au sein de la fonction associée au thread afin de modifier son comportement face au signal d'interruption. Sa signature est :

```
int pthread_setcanceltype (int t, int* ot)
```

Le premier argument correspond au nouvel état :

- `PTHREAD_CANCEL_ASYNCHRONOUS` (le thread traite rapidement de la demande)
- `PTHREAD_CANCEL_DEFERRED` (le thread retarde le traitement de la demande).

Le second argument permet de récupérer l'ancien type.

Les points d'arrêt

Les points d'annulation sont des instructions, au sein de la fonction associée au thread, dont l'exécution entraîne la vérification de l'existence de demande (et son traitement).

La liste de ces instructions est grande et est disponible dans le manuel. On y trouve notamment :

- `pthread_join`
- `pthread_testcancel`
- `printf`

Voir http://linux.die.net/man/7/pthreads

La fonction pthread_testcancel

Cette fonction permet de vérifier (de façon explicite) si une demande d'interruption a été formulée (par rapport aux autres fonctions listées précédemment, où cette vérification est implicite).

Sa signature est :

```
void pthread_testcancel (void)
```

Exemple

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

void *fct (void *pt)
 {
  int i;

  pthread_setcancelstate (PTHREAD_CANCEL_ENABLE, NULL);
  pthread_setcanceltype  (PTHREAD_CANCEL_ASYNCHRONOUS, NULL);

  for (i=1; 1; i++)
   {
    printf ("Le thread a execute %2d tours\n", i);
   sleep (1);
   }
 }
```

Exemple

```
int main (int c, char *v[])
 {
  pthread_t thread;
  int       val = 10;
  int*      res;

  pthread_create (&thread, NULL, fct, &val);

  sleep (10);
  pthread_cancel (thread);

  printf ("Le programme s'arrete\n");

  return 0;
 }
```

Exemple

On compile, on exécute, et on obtient le résultat ci-dessous :

```
emmanuel@emmanuel-VirtualBox:/media/sf_C_DRIVE/TRAVAIL/Cours/2012/3IC - Informatique concurrente$ gcc exemple4ThreadPOSIX.c -lpthread -o exemple4ThreadPOSIX
emmanuel@emmanuel-VirtualBox:/media/sf_C_DRIVE/TRAVAIL/Cours/2012/3IC - Informatique concurrente$ ./exemple4ThreadPOSIX
Le thread a execute  1 tours
Le thread a execute  2 tours
Le thread a execute  3 tours
Le thread a execute  4 tours
Le thread a execute  5 tours
Le thread a execute  6 tours
Le thread a execute  7 tours
Le thread a execute  8 tours
Le thread a execute  9 tours
Le thread a execute 10 tours
Le programme s'arrete
emmanuel@emmanuel-VirtualBox:/media/sf_C_DRIVE/TRAVAIL/Cours/2012/3IC - Informatique concurrente$
```

Les sémaphores

La bibliothèque semaphore.h

La gestion des sémaphores pour les threads POSIX est plus simple que celle utilisées pour les processus lourds (objet IPC).

Les fonctions dont nous avons besoin sont groupées dans la bibliothèques semaphore.h à inclure au début du programme.

La fonction sem_init

Comme son nom le suggère, cette fonction permet de créer un sémaphore. Sa signature est la suivante :

```
int sem_init (sem_t *s, int p, unsigned int v)
```

Le premier argument est un pointeur sur la structure permettant de gérer le sémaphore.

Le deuxième argument permet d'indiquer si le sémaphore est partagé par les threads d'un seul processus (0) ou de plusieurs processus (1). Dans le second cas, s désigne une case dans une zone de mémoire partagée.

Le dernier argument est la valeur de départ du compteur

La fonction sem_destroy

Comme son nom l'indique, cette fonction permet de détruire un sémaphore afin de librérer la mémoire occupée par la structure de type sem_t.

Sa signature est la suivante :

```
int sem_destroy (sem_t *s)
```

Remarque : si un thread était bloquée sur ce sémaphore, on obtient un résultat indéfinie.

La fonction sem_getvalue

Cette fonction permet de récupérer la valeur du compteur

Sa signature est la suivante :

```
int sem_getvalue (sem_t* sem, int * sval)
```

Le premier argument est le sémaphore dont on veut récupérer la valeur du compteur.

Le second argument est un pointeur sur un entier permettant de stocker cette valeur.

La fonction sem_wait

Cette fonction implémente la fonction « P » des sémaphores de Djikstra. Elle permet :

- de décrémenter le compteur ;
- de bloquer, le cas échéant, le thread tant que le compteur a une valeur nulle ou négative.

Sa signature est la suivante :

```
int sem_wait(sem_t * sem)
```

Une version non bloquante, appelée `sem_trywait`, met `errno` à `EAGAIN` si le compteur est nul ou négatif

La fonction sem_post

Cette fonction implémente la fonction « V » des sémaphores de Djikstra. Elle permet :

- d'incrémenter le compteur ;
- de débloquer, le cas échéant, un thread si le compteur passe d'une valeur nulle à une valeur positive. Elle

Sa signature est la suivante :

```
int sem_post(sem_t * sem)
```