Théorie et pratique de la concurrence – Master 1 Informatique
TP 3 : Variables de condition en C

Les variables de condition

La bibliothèque `Pthreads` propose également un autre outil de synchronisation entre processus que l'on appelles les variables de condition. Une variable de condition est déclarée comme suit :

```
pthread_cont_t (varcond);
```

Pour l'initialiser, on procède ainsi :

```
pthread_cond_init (&vacond,NULL);
```

Pour utiliser les variables conditionnelles, on dispose de trois fonctions qui doivent être appelées au sein d'une section critique créée grâce à un *mutex*.

La première fonction `pthread_cond_wait` est exécutée quand le thread appelant veut se mettre à "dormir", en attendant qu'une certaine condition sur l'état du programme devient vraie. Cette fonction prend comme argument un verrou supposé à être deja verrouillé. Elle va en meme temps (de manière atomique) libérer le verrou et mettre le thread appelant à "dormir". Quand le thread appelant s'éveille (à cause d'un signal envoyé par un autre thread – voir ci-dessous) la fonction `pthread_cond_wait` va reprendre le verrou. Ce comportement particulier prévient certaines conditions de course (des access non-protégées à des variables partagées). Un code utilisant cette fonction aura donc l'allure suivante :

```
pthread_mutex_lock(&verrou);
pthread_cond_wait (&varcond,&verrou);
pthread_mutex_unlock(&verrou);
```

Pour réveiller un *thread* bloqué sur une variable de condition, un autre thread peut faire :

```
pthread_cond_signal(&varcond);
```

Là aussi il est **recommandé** d'appeler cette fonction au sein d'une section critique protégée par le *mutex* utilisé par le thread en attente (dans l'exemple précédent il s'agit du *mutex* `verrou`). On peut aussi réveiller tous les *threads* en attente grâce à la fonction :

```
pthread_cond_broadcast(&verrou);
```

Notez que les variables de conditions ne peuvent pas être utilisées toutes seules. Il faut les combiner avec des conditions sur l'état du programme qui en gros assurent que l'appel à `pthread_cond_wait` se fait toujours avant l'appel à `pthread_cond_signal`. Par exemple, le code suivant peut faire qu'un des threads reste toujours bloqué (pour le moment ignorez les commentaires) :

```
void foo () {
  pthread_mutex_lock(&verrou);
  // while (done == 0)
  pthread_cond_wait (&varcond,&verrou);
  pthread_mutex_unlock(&verrou);
}

void bar () {
  pthread_mutex_lock(&verrou);
  // done = 1;
  pthread_cond_signal (&varcond);
```

```
  pthread_mutex_unlock(&verrou);
}

int main() {
  pthread_t t1,t2;
  pthread_create(&t1, NULL, foo, NULL);
  pthread_create(&t2, NULL, bar, NULL);
}
```

Le deuxième thread (qui exécute `bar`) peut s'exécuté entièrement avant le premier (qui exécute `foo`). Donc, le premier thread ne va être jamais réveillé. En ajoutant une condition sur une variable partagée, montrée dans les commentaires ci-dessus, va exclure cette situation.

Exercices

Exercice 1: *Mémoire partagée simple*

Reprendre l'exercice 4 du Tp1 en utilisant les variables de conditions pour attendre si le flag est mis à vrai ou faux.

Exercice 2: *Problème du bus*

Nous considérons N passagers et un bus ayant C places (avec $C < N$). Le comportement du bus est le suivant :

(a) Il attend que C passagers soient montés.

(b) Il part

(c) Il attend que les C passagers soient descendus du bus pour aller de nouveau au point (a).

Le comportement d'un passager est le suivant :

(a') Il essaie de monter dans le bus, si il y a encore de la place, il peut monter sinon il doit attendre.

(b') Une fois monté dans le bus, il fait le voyage dans le bus.

(c') Il descend du bus et retourne en (a').

Représenter ce système avec des processus concurrents utilisant des verrous et des variables de condition.

Indication : Voilà une façon possible d'aborder le problème :

- Les passagers pourront à l'aide d'un compteur partagé se compter de façon à savoir combien de passagers sont montés dans le bus et combien sont descendus. (Attention à protéger l'accès à ce compteur partagé grâce à un verrou).
- Le dernier passager à remplir le bus réveillera le bus pour lui signaler qu'il est plein.
- Le dernier passager à descendre du bus réveillera le bus pour lui signaler qu'il est vide.

Remarques : Observer différentes exécutions et dire si un passager voulant monter dans le bus n'y arrive jamais. Si c'est le cas, savez-vous d'où vient ce problème ? Si ce n'est pas le cas, pouvez expliquer pourquoi ?