Theorie et pratique de la concurrence { Master 1 Informatique
TP 3 : Programmation concurrente en C

Un *thread* est un processus leger, c'est-a-dire un processus qui a ses propres compteur programme et pile d'execution, mais aucun des donnees \lourdes" associees en general avec un processus (par exemple les tables des pages memoire).

L'interface de la bibliotheque `Pthreads` a ete proposee dans les annees 1990 comme un standard POSIX (d'ou le pre xe P) pour les routines de programmation multi-*thread* en C. Di erentes implementations de ce standard sont actuellement disponibles sur les systemes de type UNIX. Cette bibliotheque contient une douzaine de fonctions pour le contrôle des *threads* et pour la synchronisation.

Creation et terminaison de *threads*

Pour utiliser la bibliotheque `Pthreads` dans un programme C il faut, premierement, inclure son chier de declarations standard :

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

Deuxiemement, il faut declarer un ou plusieurs descripteurs de *threads* comme suit :

```
pthread_t pid, cid;   /* descripteurs de thread */
```

Finalement, on cree les *threads* comme suit :

```
pthread_create(&pid, NULL, start_func, arg);
```

Le premier argument est l'adresse d'un descripteur de *thread*, le deuxieme argument (que nous mettrons toujours a `NULL` dans ces TP) est un attribut de thread de type `pthread_attr_t *` (ces attributs servent par exemple a de nir la taille de la pile ou encore a imposer une politique d'ordonnacement), le troisieme argument est le nom de la fonction a executer au lancement du thread, il doit être de type `void *startfunc(void *arg)` et le quatrieme argument correspond a l'argument passe a la fonction `startfunc` au moment de l'appel, il doit être de type `void *` (cet argument peut aussi être `NULL`). Si la creation est e ectuee correctement, la fonction renvoie 0, sinon un code d'erreur. Au moment de l'appel a la fonction `pthread_create`, le *thread* correspondant est lance et peut executer la fonction `start_func`.

Un *thread* se termine en appelant :

```
pthread_exit(value);
```

ou l'argument est de type `void *` et peut être egal a `NULL`.

Un processus parent peut attendre la terminaison d'un ls en e ectuant :

```
pthread_join(pid, value_ptr);
```

ou le deuxieme argument est de type `void **` et correspond a une adresse pour le stockage de la valeur de retour du *thread* (avec `pthread_exit`).

Semaphores

Les *threads* peuvent communiquer entre eux par les variables globales. Ils peuvent se synchroniser par attente active, semaphores, verrous et variables conditionnelles. On considere ici que les semaphores.

Pour utiliser les semaphores il faut, premierement, inclure le chier de declarations standard :

```
#include <semaphore.h>
```

Deuxiemement, declarer un descripteur de semaphore global aux fonctions executees par les *threads* qui l'utiliseront :

```
sem_t sema;
```

Troisiemement, il faut initialiser les descripteurs comme suit :

```
sem_init(&sema,0, 1);
```

ou le troisieme argument correspond a la valeur initiale du compteur du semaphore (le deuxieme argument mis a 0 signi e que le semaphore peut-être partage par tous les threads du processus).

Finalement, les operations P et V sont implementees par les fonctions `sem_wait` respectivement `sem_post`. Donc, une section critique peut être implementee comme suit :

```
sem_wait(&sema);          /* P(sema) */
/* section critique */
sem_post(&sema);          /* V(sema) */
```

D'autres fonctions sont disponibles pour les semaphores, consultez le **man**.

IMPORTANT : Pour compiler vos programmes, n'oubliez pas de mettre l'option `-lpthread`

Exercices

Exercice 1: Programmer en C le probleme du producteur-consommateur avec un tampon unitaire.

- Le producteur produit une suite d'entiers terminee par 0.
- Le consommateur calcule et renvoie au parent la somme des entiers consommes.
- Le processus parent attend la terminaison de ses ls et a che la somme calculee.

Pour cet exercice, le producteur et le consommateur communiqueront a l'aide d'un tampon unitaire dont l'acces se fera en exclusion mutuelle.

Exercice 2: *Diffusion atomique*

Soit un systeme avec un processus producteur et n processus consommateurs qui communiquent via un tampon ayant b cases. Le producteur depose des messages dans ce tampon et les consommateurs les prennent. Chaque message depose par le producteur doit être pris par *tous les* n consommateurs. En plus, chaque consommateur doit prendre les messages dans l'ordre dans lequel ils ont ete deposes. Toutefois, des consommateurs di erents peuvent prendre les messages a des moments di erents. Par exemple, un consommateur peut prendre jusqu'a b messages avant un autre, si ce dernier est lent.

Programmez en C une solution a ce probleme en utilisant les semaphores.

Exercice 3: *Salle de bain unisexe*

Supposez qu'on dispose d'une seule salle de bain qui peut être utilisee par des hommes ou des femmes mais pas au même moment. Programmez une solution a ce probleme. Il faut permettre un nombre quelconque d'hommes OU de femmes être dans la salle de bain au même moment. La solution doit assurer l'exclusion mutuelle demandee et l'absence d'inter-blocage. Modi ez votre solution pour permettre au plus 4 personnes (du même sexe) en même temps.