TP/TD 7 : Les threads

1 Rappel : Processus vs. Threads

Jusqu'à présent, nous avons travaillé avec des processus "classiques", créés avec l'appel système `fork` (Cas `a.` de la Figure 1). Nous allons maintenant travailler avec des threads, appelés aussi "processus légers". Plusieurs threads peuvent être créés au sein d'un même processus (Cas `b.` de la Figure 1). De ce fait, ils partagent le même espace mémoire, mais également le même pid, les descripteurs de fichiers ouvertis, ... Chaque thread a cependant son propre identifiant, son propre pointeur de pile, masque de signaux, errno.
Nous allons au cours de ce TP voir les particularités de la programmation des threads, notamment les difficultés de synchronisation.

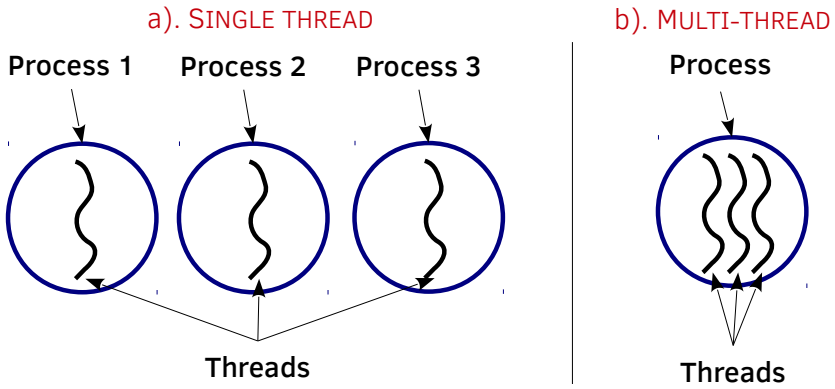

Figure 1: Processus vs. Threads

a). SINGLE THREAD: Trois processus qui contiennent chacun 1 thread

b). MULTI-THREAD: Un processus qui contient 3 threads

2 Hello Thread!

Question 1

Créez un fichier hello.c qui contient un `main` et une fonction `helloThread`. Vous devez écrire un programme qui crée 10 threads, qui exécutent chacun la fonction `helloThread`, qui a pour seul objectif d'écrire sur la sortie standard un entier correspondant au numéro du thread (dans l'ordre de création). Cet entier devra donc être passé en paramètre au thread.

Qu'observez vous lors de l'exécution ?

Notes: La fonction de création d'un thread est `pthread_create`, le manuel pourra vous être utile.
Vous aurez besoin de compiler avec `gcc -pthread -o hello <hello.c>`

Question 2

En plus d'un entier correspondant au numéro du thread, nous voulons transmettre à chaque thread un caractère faisant partie d'une chaîne. La chaîne de caractère à transmettre sera initialisée dans le main. Comment procéder pour faire passer au thread plus d'un argument ?

Question 3

Et si nous voulons que les threads s'exécutent de façon séquentielle dans l'ordre de création.
Que faut il rajouter ? Testez.

3 Gare à l'impact...

Certains trains ou tramways doivent parfois emprunter des voies uniques bidirectionnelles. Par exemple, la ligne 2 du tramway de la ville de Valenciennes en comporte 13km. Cela signifie que plusieurs tramways peuvent

emprunter cette voie s'ils circulent dans le même sens, mais quand une rame est engagée sur une portion de voie unique, aucun tramway ne peut s'engager sur cette même portion, et doit attendre que la voie soit libre.

Imaginons la situation suivante : sur le trajet entre les stations A et B, il existe une portion de voie unique nommée V. Les extrémités de cette voie sont notées V_OUEST et V_EST. La circulation entre A et V_OUEST et entre B et V_EST s'effectue à double sens. Vous devez gérer la circulation alternée sur la portion de voie unique V. Deux types de tramways peuvent exister : des trams A->B, et des trams B->A.

Le tram A->B aura donc le comportement suivant:
- Parcours A - V_OUEST
- Entrée sur la voie unique par V_OUEST
- Parcours V_OUEST - V_EST
- Sortie de la voie unique par V_EST
- Parcours V_EST - B

Et inversement pour le tram B->A.

Question 1

Vous devez implémentez une solution pour la gestion de cette circulation. Chaque type de train sera représenté par un type de thread. La voie unique représente une ressource partagée. Combien de mutex sont nécessaires à sa bonne gestion ?

Note: Les temps de parcours seront implémentés par des temps de pause (sleep) dont la durée est choisie de façon aléatoire.

Question 2

Bonus: Sans gestion particulière de priorité, que peut-il se passer ? Que proposez-vous pour gérer au mieux la circulation alternée ?

4 Vies parallèles

Le but de cet exercice est de paralléliser un jeu de la vie. Télécharger le fichier `life.c` sur le site des TDs.

Question 1

Examinez le code. Quelle partie du code peut-être facilement parallélisable?

Question 2

Modifier la fonction `board_transition` pour qu'elle utilise deux *threads* qui s'occupent chacun de la moitié du tableau. A quel moment faut-il utiliser des mutex?

Question 3

Bonus: Utilisez n^2 threads.