Système d'Exploitation Unix
TP
Les PThreads

Tous les programmes devront être développés avec passage de leurs éventuels paramètres à la fonction `main (int argc, char *argv [])`. Les valeurs de retour des appels aux primitives devront être testées et les messages d'erreurs affiches avec `perror`. Les messages d'erreurs à destination de l'utilisateur se feront sur le fichier standard des erreurs `stderr`.

Question 1 Création de threads

Ecrivez un programme qui crée n threads. Chaque thread exécute une boucle suffisamment longue pour permettre l'observation et dans laquelle il écrit à chaque itération:

`Bonjour, je suis le thread` *`tid`*

Le thread 1 affiche son message au debut de la ligne, le thread 2 à une tabulation du début de la ligne, le thread 3 à deux tabulations du début de la ligne, etc

Fonctions utiles:

- *pthread_create, pthread_exit (): cf. cours.*

Question 2 Synchronisation des threads

a) Ecrire un programme qui crée :

- n threads (n<=26) qui écrivent un caractère dans toutes les entrées d'un même tableau. Le thread écrivain 1 écrit le caractère A, le thread écrivain 2 écrit le caractère B, etc.
  Chaque thread écrivain est créé "détaché". Il reçoit en paramètre son numéro d'ordre de création
- un thread lecteur qui affiche le contenu complet du tableau et qui compte le nombre de fois où un caractère apparaît dans le tableau. Ce thread lecteur du tableau est créé "joignable" et sera attendu par le thread `main`.
  Il reçoit en paramètre le nombre de fois où il doit afficher le contenu du tableau.

  Lorsque le thread lecteur est terminé, le thread `main` affiche le nombre d'apparitions de chaque caractère constatées par le thread lecteur.

Compilation: `gcc nom.c -pthread -o nom`

Appel: `./nom 3 5`

Résultat pour un tableau de 75 caractères:

```
DEBUT MAIN
CBBCAAAACCAACBBBBAAACBCCCCCBBBBBCCAABBBBBACCCCAAAAABBBCCCCCBBAAAACCCCCCCBCC
CCBAACCAAAAACCBBBBBBBACBBBBCAAACCBACCCCCACCCCAAAABBBBAAAAACCAAAAAAAABABBBCC
CAAAAAAAACBBBBAAAAAACCCCAAAACCCABBBBBAAAABBBBACCCCCABCCCAAAABBAAAAAAAAACBBB
CBCAAAAABBBACCCCCBBBCCCAAAAABBCCCCCABBBBBCCAABBBBAACCCCAACCCCCCAAAACAAAAAAA
CCCAAAACCCCCCAAACCCCCCCCCCCCCCCCCCCAACCCCAACCCCAAACCCCCCCCAAAAACAAAAAAAAACC
A: 147
B: 81
C: 147
FIN MAIN
```

b) Procédez à plusieurs exécutions de : `./nom 3 5` . Expliquez ce que vous constatez.

c) Modifiez le programme précédent afin que

1. le thread lecteur
   - affiche chaque caractère lu du tableau et le remplace par un espace
   - ne s'exécute que lorsque le tableau ne contient que des caractères différents de l'espace
2. chaque thread écrivain :
   - soit le seul à écrire son caractère dans le tableau **au moment où il écrit ce caractère**.
   - n'écrive son caractère que dans les cases du tableau contenant un espace
   - ne s'exécute que lorsque le tableau a été rempli d'espaces par le thread lecteur.
   - Modifiez le programme précédent afin que

d) Expliquez pourquoi ce programme ne permet pas tout le parallélisme possible entre les threads écrivains.

e) Proposez et codez une solution permettant tout le parallélisme possible entre les threads écrivains.

Fonctions utiles:

pthread_create(), pthread_exit(), pthread_attr_init(), pthread_attr_setdetachstate(), pthread_join(), pthread_mutex_lock(), pthread_mutex_unlock(),pthread_cond_wait(), pthread_cond_signal(),pthread_cond_broadcast()