Ecole Supérieure Privée d'Ingénierie et de Technologies	**Examen** **Semestre : 2** **Session : PRINCIPALE**
Module : Système d'exploitation avancé (SEA2) **Enseignant(s)** : UP Système **Classe(s) :** 4EME ANNEE	**Documents autorisés** : NON **Nombre de pages** : 2 **Date** : 30/05/2022 **Heure 08h30 Durée** : 01h30

Exercice 1 :Thread (8 pts)

1. Ecrire un programme ayant le comportement suivant :
   - Des threads sont créés (leur nombre étant passé en paramètre lors du lancement du programme) ;
   - Chaque thread affiche un message (par exemple hello world !).
   - Le thread principal attend la terminaison des différents threads créés.
2. Modifier le programme de la question précédente pour que chaque thread affiche :
   - son PID(avec **getpid()**) ;
   - La valeur opaque retournée par pthread_self, par exemple avec :

printf("%p \n", (void *) pthread_self()) ;

```
$./thread  5
Hello world! pid=22755
pthread self=0x7fef3c0ab700
Hello world! pid=22755
pthread self=0x7fef3c8ac700
Hello world! pid=22755
pthread self=0x7fef3d0ad700
Hello world! pid=22755
pthread self=0x7fef3b8aa700
Hello world! pid=22755
pthread self=0x7fef3b0a9700
Fin de l'attente
```

3. Modifier le programme de la question précédente pour passer son numéro d'ordre à chaque thread. Chaque thread doit ensuite l'afficher. (Pensez à faire un changement dans la fonction « pthread_create »)

```
$. /thread 5
Hello world! i=1
Hello world! i=4
Hello world! i=2
Hello world! i=3
Hello world ! i=0
Fin de l'attente
```

4. Déclarez une variable globale **somme** initialisée à 0. Chaque thread doit, dans une boucle, ajouter 1 000 000 fois son numéro d'ordre à cette variable globale (on veut bien faire 1 000 000 additions par thread, pas juste une). Affichez la valeur obtenue après la terminaison de tous les threads.

Vous trouverez dans ce lien la correction

https://members.loria.fr/lnussbaum/RS/06-tp3-threads-correction.pdf

Exercice 2 : Attente active (4 pts)

On vous propose ci-joint une solution au problème d'exclusion mutuelle par variables partagées (attente active). La solution utilise une variable ***tour*** et deux variables ***Di*** {i=1,2) (Une variable ***Di*** par processus, ***D0 = faux et D1 = faux***):

NB : La proposition est écrite pour le cas de deux processus P0 et P1

Initialisation
tour= 0
D0, D1 = faux

```
      tant_que vrai faire
      |  actions avant section critique
1:    |  Di ← vrai
2:    |  tour ← 1-i
3:    |  tant_que (D(1-i) et (tour = 1-i)) faire
      |  |  rien
4:    |  section critique i
5:    |  Di ← faux
      |  actions après section critique
```

Questions :

1- Vérifier si les quatre spécifications de l'exclusion mutuelle sont réalisées ?
   Les 4 conditions sont vérifiées (solution de Peterson)
2- Combien de passages en section critique de P(1-i), Pi doit-il attendre au plus lorsqu'il a demandé à passer en section critique ?
   1

Exercice 3 : Attente inactive (8 pts)

Un train est constitué de N wagons. Chaque wagon est doté d'une porte automatique. Devant chaque porte, se trouve un bouton qui permet au passager de l'ouvrir en appuyant dessus. Pour la sécurité des passagers, les règles suivantes doivent être strictement respectées :
- Un passager ne peut pas ouvrir une porte si le train est en marche.
- Le train ne peut pas démarrer si au moins l'une des portes est ouverte.
- Une porte se ferme automatiquement après 30 secondes de son ouverture.

Le programme qui gère l'ouverture/fermeture des portes s'exécute sur un système d'exploitation multitâche embarqué dans le système informatique du train. Il est composé des fonctions suivantes :

- **démarrer_train ()** : fonction exécutée par le conducteur de train. Son rôle est de démarrer le train.
- **arrêter_train()** : fonction exécutée par le conducteur de train. Son rôle est d'arrêter le train.
- **ouvrir_porte()** : fonction exécutée par le passager en appuyant sur un bouton. Son rôle est d'ouvrir une porte.
- **fermer_auto_porte()** : fonction exécutée automatiquement 30 secondes après l'ouverture d'une porte.

Ces fonctions s'exécutent simultanément et modifient de façon concurrentielle les variables **état_train** et **nb_portes_ouvertes** :

- **état_train = 'M'** si le train est en marche, **état_train = 'A'** si le train est en arrêt.
- **nb_portes_ouvertes** : nombre de portes ouvertes à un instant donné.

Initialement, **état_train ='A'** et **nb_portes_ouvertes=0**.

1- A quel exemple vu dans le cours correspond ce type de problème.

   Problème point de rendez-vous. (Exercice 1 TD)

2- En utilisant les sémaphores, écrire les codes des fonctions **démarrer_train()**, **arrêter_train()**, **ouvrir_porte()** et **fermer_auto_porte()**.

```
etat_train= 'A'
semaphore  mutex=1 ;
semaphore   syn=0 ;
nombre_portes_ouvertes=0 ;
```

Démarrer_train ()

```
{
down (mutex)
If (nombre_portes_ouvertes > 0)
{
    up (mutex) ;
    down(syn) ;
}
else
{
Démarrer.....
etat_train = ''M''
 Up (mutex) ;
}
```

Fermer_auto_porte

```
{
Down (mutex)
Nb_portes_ouverts = Nb_portes_ouvertes -1 ;
 If (nombre_portes_ouvertes == 0)
{
Up (syn)
}
Up(mutex)
}
```

Arrêter _train ()

```
{
Down (mutex)
Arrêter
etat_train= 'A' ;
Up(mutex)
}
```

Ouvrir_portes

```
{
Down ( mutex)
If (etat_train == 'A') {
Nb_portes_ouvertes =  Nb_portes_ouvertes +1 ;
}
Up (mutex)
}
```