Programmation Temps Réel - TP2
Interblocage, modèle producteurs/consommateurs et variables conditionnelles

A - Consignes générales

Réalisation des exercices :

- Les programmes sont réalisés en langage C et doivent tourner sous Linux.
- Privilégier un style clair et lisible
- Nommer les fonctions et les variables avec des noms appropriés
- Commentez votre code
- Soignez votre indentation
- **Chaque exercice doit faire l'objet d'un programme en C séparé (fichier .c).**
- **Les réponses aux questions doivent être placées sous forme de commentaires au début du fichier.c** de l'exercice correspondant ou bien dans un fichier texte.
- **La compilation ne doit retourner ni erreur, ni warning.**

Remise des exercices :

Nommer les fichiers pour indiquer à quel exercice ils correspondent (exo1.c, exo3.txt, etc...)

Déposer les fichiers sur : https://moodle.univ-paris8.fr/moodle/course/view.php?id=1373 code 'ptr'

B - Énoncés des exercices

Exercice 1 - Le diner des philosophes et l'interblocage

Récupérez le fichier exo1_philo.c sur la page Moodle du cours.

- Examinez le code, compilez-le et lancez-le.
- Observez que pour le moment plusieurs philosophes peuvent utiliser la même fourchette (représentée par un sémaphore) en même temps.
- Ajouter les instructions nécessaires pour attendre une fourchette ou relâcher une fourchette dans les fonctions **forkWait()** et **forkPost()** de manière à ce qu'une fourchette ne puisse être utilisée que par un seul philosophe à la fois.
- Re-compilez le code et lancez-le : vous devez obtenir un inter-blocage

Faites un copier-coller des commandes saisies en console pour compiler et lancer le programme, ainsi que d'un exemple de retour du programme quand un interblocage se produit. Placer ces lignes en commentaire à la fin de votre fichier.

Exercice 2 - Résoudre un problème d'interblocage 1/2

Reprenez le code du fichier précédent et tentez de résoudre le problème de l'interblocage en limitant le nombre de philosophes qui mangent en même temps grâce au sémaphore `numEating`.

Question : Quel est la valeur maximale de `NUM_EATING` permettant d'éviter l'interblocage ?

Exercice 3 - Résoudre un problème d'interblocage 2/2

Reprenez le code de l'exercice 1 et implémentez une autre solution au problème de l'interblocage empêchant l'attente circulaire. Ajouter des commentaires pour signaler et expliquer les instructions que vous avez ajoutées/modifiées.

Exercice 4 - Le modèle producteur / consommateur

Voici un rappel de la structure d'un programme utilisant le modèle producteur / consommateur :

```
mutex m = 1;         // protège l'accès au tampon
semaphore plein = 0; // présence d'un tampon contenant un message ?
semaphore vide = n;  // limite à n le nombre de tampons
buf_type tampon[n];
producteur() {
    buf_type *tp, *new;
    while(1){
        new = produire();      // création un nouvel élément
```

```
        P(vide);                // y a t-il un tampon libre ?
        P(m);
        tp = obtenir(tampon);    // obtention du tampon
        V(m);
        copier(new, tp);
        P(m);
        placer(tp, tampon);     // met tampon dans liste des tampons
        V(m);
        V(plein);               // signale la présence d'un tampon plein
    }
}
consommateur() {
    buf type *tp;
    while(1){
        P(plein);                // y a t-il un message à consommer ?  // (*)
        P(m);                                                         // (*)
        tp = obtenir(tampon);    // obtention du tampon plein
        V(m);
        consommer(tp);
        P(m);
        placer(tp, tampon);      // met tampon dans liste des tampons
        V(m);
        V(vide);                 // signale le libération du tampon consommé
    }
}
```

Question : Que se passe-t-il si l'on inverse P(plein) et P(sem) signalés par (*) dans le consommateur ?

Décrivez en détails ce qui se produit.

Placez votre réponse dans un fichier texte.

Exercice 5 - Modèle producteur / consommateur, implémentation 1/2

Récupérez le fichier exo5_prod_cons.c sur la page Moodle du cours.

- Examinez le code
- Commentez les fonctions parent et enfant. Expliquez ce qu'elles font.
- Compilez-le programme et lancez-le.

Faites un copier-coller des commandes saisies en console pour compiler et lancer le programme, ainsi que d'un exemple de retour du programme. Placer ces lignes en commentaire dans votre fichier.

Question : En comparant ce programme au modèle producteur / consommateur précédent, expliquer ce qui empêche ce programme de fonctionner correctement.

Exercice 6 - Modèle producteur / consommateur, implémentation 2/2

Ajouter les instructions nécessaires pour que le programme précédent fonctionne comme le modèle producteur/consommateur donné à l'exercices 3.

Question : Le programme fonctionne-t-il avec plusieurs producteurs et plusieurs consommateurs ?

Si ce n'est pas le cas pour votre programme, donnez un exemple du résultat obtenu en console et expliquer l'origine du problème.

Exercice 7 - Attente active

Récupérez le fichier exo7_var_cond.c sur la page Moodle du cours.

- Examinez le code, compilez-le et lancez-le.
- Commentez les fonctions parent et enfant. Expliquez ce qu'elles font.
- Un des threads utilise une attente active, en ajoutant des commentaires au code, expliquez ce que c'est et où cela est réalisé.

Faites un copier-coller des commandes saisies en console pour compiler et lancer le programme, ainsi que d'un exemple de retour du programme. Placer ces lignes en commentaire à la fin de votre fichier.

Exercice 8 - Variables conditionnelles

Le programme précédent est maladroit. Eliminez l'attente active et remplacez-la par des instructions utilisant la variable conditionnelle « gateaux_prets ». Faites en sorte que le parent prévienne tous les enfants quand une fournée de gâteaux est prête.