HMIN101 - Réseaux et Communications - TD/TP 2
IPC : Mémoires partagées et ensembles de sémaphores

1 Gestion d'un service avec une mémoire partagée

Remaque : L'implémentation des algorithmes de cet exercice est optionnelle. Celle des exercices suivants est prioritaire.

On veut mettre en place un système de communication particulier entre un chef de service et les employés de ce service. Ce système consiste à permettre au chef de service d'envoyer des messages (instructions, actualité, blagues, etc.) aux employés pendant les heures de travail. Sauf que le chef de service ne dispose que d'un espace mémoire limité, dans lequel il est possible d'écrire un message. Par conséquent, à chaque fois qu'un message est écrit, le précédent est écrasé. Le chef de service doit donc s'assurer que l'ensemble des employés ait bien lu un message avant d'écrire le suivant.

Pour réaliser cette application nous allons utiliser un segment de mémoire partagée qui contiendra un message écrit par un "processus chef de service". Nous utiliserons aussi des sémaphores. Nous supposons que chaque poste employé est relié à un processus (qu'on appellera : processus employé, ou employé tout court) lui permettant de lire un message depuis la mémoire partagée.

Un segment de mémoire partagée contient donc un message posté par le chef de service et peut être lu (nous supposons une seule lecture) par tous les employés. Le rôle de chaque processus employé (respectivement, processus chef de service) est de lire (resp. écrire) un message. Aussi, après chaque lecture/écriture d'un message, un processus se termine. Il faudra donc le relancer pour chaque lecture/écriture.

Dans un premier temps, nous supposons un seul employé.

1. Montrer qu'il est nécessaire de mettre en place une exclusion mutuelle entre le processus chef de service et l'employé.
2. Proposer une solution à l'aide de sémaphores et écrire le schéma algorithmique des processus employé et chef de service.
3. Implémenter votre alghorithme et assurez vous de son bon fonctionnement.

   Nous supposons maintenant qu'il y a n employés, tous doivent lire chaque message envoyé par le chef de service.

4. Modifier l'algorithme précédent pour tenir compte des n employés. Implémenter le résultat.
5. Que ce passe-t-il si un $n + 1^{eme}$ processus employé est lancé avant la fin du processus chef de service ? Si problème, comment le résoudre ?

2 Rendez-Vous

Il est souvent nécessaire de réaliser un rendez-vous entre processus. Pour lancer un jeu à plusieurs joueurs par exemple, pour synchroniser des calculs, etc.

Proposer une solution utilisant un sémaphore et permettant à n processus d'attendre jusqu'à ce que tous les processus soient présents et qu'ils soient arrivés à un point déterminé dit *point de rendez-vous* de leur code, ceci avant de poursuivre leur exécution.

Implémenter cette solution, en affichant la valeur du sémaphore à chaque arrivée d'un processus au point de rendez-vous (utiliser *semctl()*).

3 Traitement synchronisé

On envisage ici le traitement parallèle d'une image par plusieurs processus, chacun ayant un rôle déterminé. Par exemple, un processus pourrait faire du lissage, un autre des transformations de couleurs, ou de l'anti-crénelage, etc. D'une façon générale, chaque processus travaille sur un ensemble de points (pixels) de l'image. Appelons ces ensembles *zones*. On peut donc considérer l'image comme une suite de *zones* ordonnées.

Le travail doit se faire de la manière suivante :

- chaque processus doit traiter, dans l'ordre, toutes les zones de l'image,
- avec garantie d'exclusivité : aucun autre processus ne doit accéder en même temps à la zone en cours de traitement par un processus donné,
- les différents traitements doivent se faire dans un ordre déterminé entre les processus : sur toute zone, le processus P_1 doit passer en premier, puis P_2, etc.

On se limite d'abord à deux processus. Proposer une solution permettant un fonctionnement correct, sachant que l'image est stockée dans un segment de mémoire partagée.

Pour passer à trois processus (et plus) que peut-on proposer ?

Implémenter progressivement vos algorithmes, en simulant un temps de travail aléatoire pour chaque traitement. Générer un temps de travail suffisamment long, de sorte à pouvoir corriger les éventuelles erreurs et à montrer que la protection mise en place fonctionne.