Programmation Système

Travaux Pratiques (8), Licence 2 Informatique

Les IPC SYSTEM V - Les ensembles de sémaphores

Les exercices suivants ont pour but de vous familiariser avec les IPC (Inter-Process Communication) SYSTEM V. Cette partie se consacre aux ensembles de sémaphores.
Il vous est recommandé de consulter les pages man des primitives relatives aux IPC pour de plus amples informations sur leur syntaxe, leur sémantique et les éventuelles options qu'elles offrent.

Les instructions des exercices se repèrent par des icônes, qui sont les suivantes :

	Information	Information concernant l'usage ou le rôle d'une commande, par exemple. Dans certains cas, il s'agit d'une information sur ce que vous êtes en train de faire ou sur ce qui se passe.
	Exemple	Exemple d'utilisation.
	Contrôle	Vérifier le résultat d'une (ou plusieurs) action(s).
	Action	Effectuer la ou les action(s) décrite(s).
	Question	Questions auxquelles vous devez répondre.

De plus, un `texte en police courier` correspond soit à une sortie écran soit à des noms spécifiques (menus, fenêtre, icône, processus, commandes…).
Un **texte en police times gras** correspond à ce que l'utilisateur doit introduire comme valeur de paramètre, ou encore, est utilisé pour attirer l'attention de l'utilisateur.

Création d'un ensemble de sémaphores

Les ensembles de sémaphores constituent un mécanisme de synchronisation des processus. La primitive `int semget(key_t cle, int nsems, int flags);` permet de créer un ensemble de `nsems` sémaphores associé à la clé `cle`.

Écrire un programme qui crée un ensemble de sémaphore en spécifiant la clé privée (`IPC_PRIVATE`). On se limitera, dans un premier temps, à la création d'un ensemble constitué d'un seul sémaphore.

Avant la terminaison du programme, vérifier que le sémaphore a bien été créé. Utiliser pour cela la primitive `ipcs` alors que le programme est suspendu (`sleep`).

Le sémaphore créé existe-t-il encore après la terminaison du programme ? Qu'en déduisez-vous ?

Mohamed Taghelit , taghelit@univ-tours.fr

Recherche de l'identifiant d'un ensemble de sémaphores

La primitive `int semget(key_t cle, int nsems, int flags);` permet également de rechercher l'identifiant de l'ensemble de sémaphores associé à la clé `cle`.

Modifier le programme précédent de façon qu'il recherche l'identifiant du sémaphore précédemment créé.

Quel problème rencontrez-vous ? Expliquer !

Dans quel cas peut-on utiliser une clé privée ? Expliquer !

Modifier le premier programme de façon qu'il crée un sémaphore avec une clé que vous aurez construit (utiliser la fonction `ftok`).

S'assurer que le sémaphore est bien créé.

Modifier le programme précédent de façon qu'il recherche l'identifiant du sémaphore précédemment créé.

S'assurer qu'un nouvel ensemble de sémaphores n'a pas été créé.

Opérations de contrôle sur les sémaphores

Extraction et modification des caractéristiques d'un (ensemble de) sémaphore(s)

La primitive `int semctl(int sem_id, int semnum, int cmd, union semun arg);` permet l'extraction des caractéristiques d'un ensemble de sémaphores ou d'un sémaphore de l'ensemble.

Écrire un programme qui affiche certaines caractéristiques (créateur, propriétaire, …) d'un ensemble de sémaphores existant.

Faire en sorte qu'il extrait également les caractéristiques d'un sémaphore (valeur du sémaphore, nombre de processus en attente que la valeur du sémaphore augmente, …).

La primitive précédente, permet également de modifier certaines caractéristiques d'un ensemble de sémaphores existant ou d'un (ou plusieurs) sémaphore(s).

Modifier le programme précédent de façon à ce qu'il modifie le propriétaire d'un ensemble déjà existant.

Faire en sorte qu'il établisse à 1 la valeur (`semval`) du sémaphore précédemment créé.

Suppression d'un ensemble de sémaphores

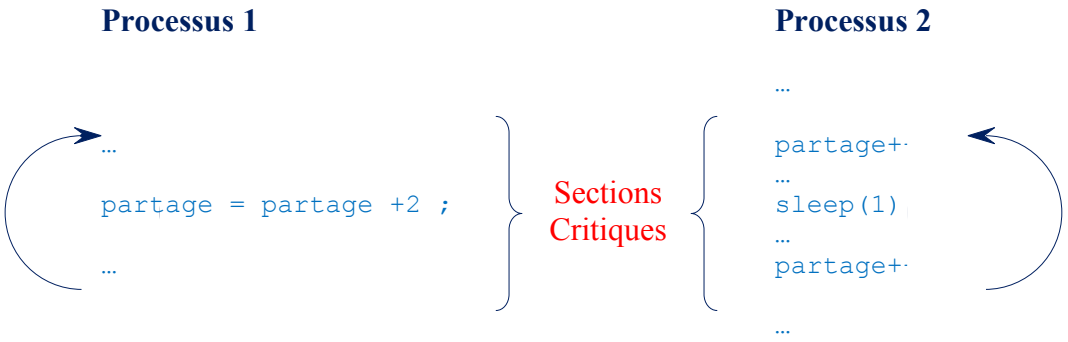
La primitive `int semctl(int sem_id, int cmd, union semun arg);` permet enfin la suppression d'un ensemble de sémaphores.

Modifier le programme précédent de façon qu'il supprime l'ensemble de sémaphores une fois terminé.

Opérations sur les sémaphores

Un sémaphore `S` est une variable à valeurs entières accessible aux travers des deux opérations `P(S)` et `V(S)` (se reporter au cours pour la sémantique de ces opérations). La primitive `int semop(int sem_id, struct sembuf *sops, unsigned nsops);` permet d'implémenter ces opérations.

Écrire deux programmes ayant chacun une section critique qu'ils doivent exécuter en exclusion mutuelle. Dans cette section critique qu'ils exécutent en boucle, il doivent accéder, à chaque itération, à une donnée partagée (par exemple, `partage`, contenue dans un segment de mémoire partagée) qu'ils incrémentent de la valeur 2. Le premier programme incrémente, dans sa section critique, la donnée en une seule fois (partage = partage + 2). Le second l'incrémente en deux fois (partage++, …, partage++), toujours dans sa même section critique (cf. schéma suivant).



Utiliser un sémaphore pour assurer l'exclusion mutuelle entre les deux processus.

S'assurer que l'exclusion mutuelle est vérifiée en affichant la valeur de la donnée partagée avant chaque sortie de la section critique dans chacun des processus. Il ne devrait jamais y avoir de valeur impaire.

Utiliser un des deux programmes pour créer et initialiser le sémaphore. Sinon, réutiliser le sémaphore créé précédemment.

Tuer l'un des processus alors qu'il est encore dans sa section critique.

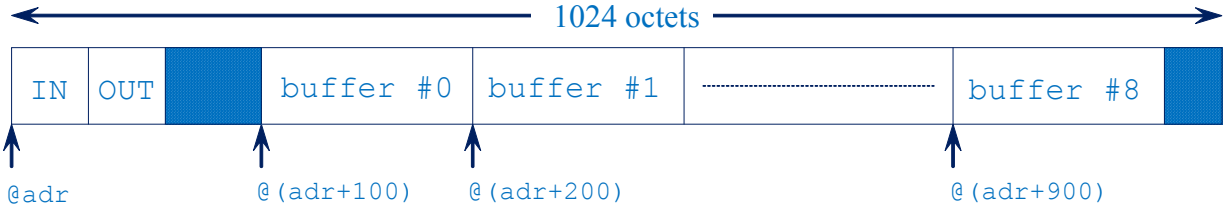
Que devient le second processus ? Quelle solution peut-on apporter ?

Réécrire les programmes de façon que le problème précédent ne se reproduise plus.

Synchronisation et exclusion mutuelle entre processus

Prenons l'exemple du modèle producteur/consommateur afin de résoudre, par l'utilisation des sémaphores, les problèmes de synchronisation et d'exclusion mutuelle entre processus.

Considérons le segment de mémoire partagée décomposé comme suit :



Soit `@adr` l'adresse de début du segment de mémoire partagée. Le segment contient `9` tampons, numérotés de `0` à `8`. Le tampon numéro `n` commençant à l'adresse `adr + (n+1)*100`. Le tampon est géré de façon circulaire, et pour cela on utilisera deux variables globales entières `IN` et `OUT`, contenues en début de segment, qui indiqueront respectivement le prochain tampon à remplir et à vider.

Un nombre `P` de processus producteurs pourront produire des messages dans les tampons. La variable entière `IN` indiquera le numéro du prochain tampon (vide) à remplir.

Un nombre `C` de processus consommateurs pourront consommer des messages des tampons. La variable entière `OUT` indiquera le numéro du prochain tampon (plein) à vider.

Il est demandé d'écrire les programmes des processus `Producteur` et `Consommateur` ainsi que le programme `Initialisation` qui aura à charge de créer le segment de mémoire partagée, les différents sémaphores utilisés ainsi que toutes les initialisations.

On pose les conditions suivantes:

- un producteur ne pourra remplir un tampon que s'il en existe un qui est vide,
- un consommateur ne pourra vider un tampon que s'il en existe un qui est plein,
- pour les producteurs, séparer l'acquisition d'un tampon de son remplissage,
- pour les consommateurs, séparer l'acquisition d'un tampon de son vidage,
- assurer l'exclusion mutuelle entre producteur et consommateur lors de l'utilisation d'un même tampon. C'est à dire, éviter qu'un producteur ne remplisse un tampon qui est en train d'être vidé par un consommateur, et inversement.

Commandes `shell` relatives aux IPC SYSTEM V

La commande `ipcs` permet d'afficher les ressources IPC existantes.

Utiliser la commande `ipcs` pour prendre connaissance des différents ensembles de sémaphores existants. Essayer quelques options permises.

La commande `ipcrm` permet de supprimer les ressources IPC existantes.

Utiliser la commande `ipcrm` pour supprimer des ensembles de sémaphores existants. Essayer quelques options permises.