Compte rendu TP Sémaphore

Hoël JALMIN, Mathieu DUMAX-VORZET, Iheb MASTOURA

Les objectifs:

Pour ce TP, nous avons implémenté les fonctions suivantes permettant de créer, supprimer et faire des opérations sur des sémaphores:

- `create`
- `down`
- `up`
- `destroy`
- `semget`
- `semctl`
- `semop`

Les structures et les tableaux:

Dans ce TP, nous avons créé 2 nouvelles structures de données :

- Semaphore**:** type contenant un champ val (pour la valeur de sémaphore) et un champ waiting_queue (pour la liste des processus en attente). La liste d'attente est gérée de manière particulière ici car nous ne pouvons pas utiliser les fonctions d'allocations. Ainsi, chaque processus possède un champ pointant sur le processus suivant de la liste d'attente (avec NULL comme valeur sentinelle) et le premier processus de la liste est pointé par le pointeur waiting_queue.

- sem_state**:** type contenant des informations sur un sémaphore actif. Pour chaque colonne, il possède un champ s (pour le sémaphore correspondant), un champ key (la clé associé à ce sémaphore) et un champ nbproc (correspondant au nombre de processus utilisant ce sémaphore).

Nous avons également créé un tableau **current_semaphores** de type sem_state afin d'avoir une table des sémaphores actifs. Sa taille est définie par la macro **SIZE_SEM_TAB**, qui a une valeur identique à **PROC_MAX** (il semble raisonnable de supposer qu'on aura pas davantage de sémaphore que de processus actifs) .

On a également un entier qui compte le nombre de sémaphore actifs, **NB_SEM**, afin de manipuler plus efficacement la table.

Les opérations de base:

Les opérations de base permettant d'éditer un sémaphore et sa valeur sont les suivantes**:**

1. **create (int n):** permet de créer un sémaphore de valeur n. En effet, on teste si la valeur de n est positive; si oui on affecte la valeur de n au champ val de sémaphore et on initialise la waiting_queue à NULL, sinon on retourne un sémaphore mort (voir destroy).

2. **down (Semaphore *sem):** On diminue de un la valeur du sémaphore et si elle devient négative (c.a.d s'il n'y plus de ressource disponible), on endort le processus correspondant avec **sleep** et on le met dans la waiting_queue du sémaphore.

3. **up (Semaphore *sem):** On incrémente de un la valeur du sémaphore, et si celle-ci est positive (c.a.d on a une nouvelle ressource de disponible et des processus en attente), on réveille un processus endormi dans la waiting_queue du sémaphore grâce à la fonction **wakeup** et on le retire de la queue.

4. **destroy (Semaphore *sem):** On détruit un sémaphore en vidant sa waiting_queue et en mettant sa valeur et sa clé à **EMPTY_SEM**.

Les appels système:

Une fois que nous avons implémenté l'abstraction des sémaphores, nous avons créé les appels système permettant de les utiliser :

1. **int semget (unsigned key):** On parcours la table des sémaphores actifs pour trouver le sémaphore associé à cette key. Si celui ci existe déjà dans le tableau, on incrémente le nombre de processus utilisant ce sémaphore puisque le processus courant se met à l'utiliser aussi. Sinon, on crée un nouveau sémaphore avec la clé donnée en paramètre, avec la valeur EMPTY_SEM indiquant qu'il n'a pas encore été initialisé. On indique qu'il y a un processus (celui qui crée le sémaphore) qui l'utilise, et on incrémente le nombre de sémaphores actifs. Cette fonction renvoie l'indice du tableau du sémaphore associé à la clé ou -1 en cas d'erreur.

2. **int sys_semctl (int semid, int cmd, int val)**: Cette fonction effectue plusieurs opérations de contrôle sur un sémaphore, selon la valeur de cmd.

- ➢ Si cmd correspond à la macro **GETVAL**, on renvoie simplement la valeur du sémaphore à l'indice semid.

- ➢ Si cmd correspond à **SETVAL**, on modifie la valeur du sémaphore; celle ci devient égale à l'entier val passé en paramètre de la fonction. On retourne 0 si l'exécution s'est bien déroulée sinon, on retourne -1.

- ➢ Si cmd correspond à la macro **IPC_RMID**, on tente de détruire le sémaphore. On vérifie d'abord si aucun autre processus que le processus courant ne l'utilise et si c'est le cas on détruit le sémaphore associé à l'indice semid et on fixe sa clé à **EMPTY_SEM** et son nbproc à zéro. Sinon, on décrémente son nbproc de un. On retourne 0 si l'exécution s'est bien déroulée sinon, on retourne -1.

3. **int sys_semop (int semid, int op)**: Permet d'incrémenter ou de décrémenter la valeur du sémaphore présent à l'id semid de la table des sémaphores, selon la valeur de op. Si op est négatif, on décrémente op fois la valeur du sémaphore. Si op est supérieur à zéro, on incrémente op fois la valeur du sémaphore. Si op est nul, on ne fait rien. Cette fonction renvoie 0 si l'exécution s'est bien déroulée, et -1 s'il y a eu un problème.