Les ressources IPC : sémaphore et mémoire partagée

José Vander Meulen, Olivier Choquet, Bernard Frank, Anthony Legrand

Avant de résoudre ces exercices, nous vous conseillons de lire le document suivant qui présente une vue d'ensemble sur les sémaphores et la mémoire partagée, et qui résume également les principaux appels systèmes liés à ces matières : "Les ressources IPC : sémaphore, mémoire partagée et files de messages"[1]

Après avoir exécuté vos programmes, il est possible que certains "ipcs" non-utilisés soient réservés sur la machine sur laquelle tournait vos programmes. Pensez à utiliser les commandes "ipcs" et "ipcrm" pour supprimer ces "ipcs".

1. Le radar de recul (mémoire partagée)

Un **radar de recul** est un système permettant au conducteur d'estimer la distance séparant son véhicule d'un obstacle[2]. Nous vous demandons d'écrire :

- un programme "radar" dont le but est de simuler un radar de recul; et
- un programme "dashboard" dont le but est de simuler un tableau de bord qui affiche les informations transmises par le radar de recul.

Pratiquement, votre programme radar écrira toutes les 3 secondes, pendant 1 minute, un nombre entier aléatoire dans une mémoire partagée. Votre programme dashboard affichera toutes les 5 secondes, pendant 1 minute, l'entier se trouvant dans la mémoire partagée.

2. Mon identité (section critique)[3]

Dans un premier temps, nous vous demandons d'écrire un programme "famille1" qui crée 2 processus fils. Chaque fils affiche 3 fois le message « je suis le fils <pid> », chaque affichage étant séparé par 1 seconde d'attente. Les fils pourraient, par exemple, afficher le texte ci-dessous.

```
je suis le fils 123
je suis le fils 124
je suis le fils 123
je suis le fils 124
je suis le fils 123
je suis le fils 124
```

[1] M. Bagein, S. Frémal, S. Mahmoudi et P. Manneback, *Les ressources IPC : sémaphore, mémoire partagée et files de messages*, Travaux pratique d'Informatique Temps Réel, UMONS/Polytech, TP 3, Année Académique 201415

[2] https://fr.wikipedia.org/wiki/Radar_de_recul, 15/03/2018

[3] M. Bagein, S. Frémal, S. Mahmoudi et P. Manneback, *Ibid*

Dans un deuxième temps, nous souhaitons empêcher l'affichage simultané des deux fils, en considérant l'accès à “stdout” comme une ressource critique. Pour ce faire, nous vous demandons d’écrire un programme “famille2” qui est une variante du programme “famille1”. Lorsqu'un fils démarre son exécution, il réserve l'accès à “stdout” et réalise l'affichage de ses 3 messages. Ensuite, il “libère” l'accès “stdout”. Le deuxième fils peut alors réaliser son affichage. Par exemple, les deux fils pourraient afficher le texte ci-dessous.

```
je suis le fils 123
je suis le fils 123
je suis le fils 123
je suis le fils 124
je suis le fils 124
je suis le fils 124
```

3. Le distributeur de tickets (mémoire partagée et section critique)

Lorsqu’une personne habitant Louvain-la-Neuve se rend à la maison communale, elle reçoit un ticket portant un numéro et va s'asseoir dans la salle d'attente. Quelques instants plus tard, le numéro de son ticket s’affiche sur un grand écran. La personne se rend alors au guichet où se trouve un employé communal.

Nous vous demandons d’écrire un programme “distributeur” et un programme “ecran”. Votre programme “distributeur” ne fait qu’une chose : il imprime sur “stdout” le numéro du prochain ticket et se termine. Votre programme “ecran” ne fait également qu’une chose : simulant le fait qu’une place s’est libérée au guichet, il affiche le prochain numéro sur le grand écran puis se termine. S’il existe un prochain numéro, “ecran” l’imprime sur “stdout”, sinon il affiche le message “`Il n’y a plus personne !`”.

Notez que les deux programmes modifient la mémoire partagée de manière adéquate. Le programme “distributeur” s'exécute entièrement chaque fois qu'un nouvel arrivant déclenche une demande de tickets. De même, votre programme “ecran” s'exécute entièrement chaque fois qu’une place se libère au guichet.

Notez que plusieurs programmes “distributeur” et plusieurs programmes “ecran” peuvent s’exécuter simultanément. Vous devez donc protéger vos accès à la mémoire partagée à l’aide de sémaphores. En plus des deux programmes, nous vous conseillons d’écrire un troisième programme “admin” dont le but est de créer et d’initialiser la mémoire partagée et les sémaphores de manière indépendante. Ce programme sera exécuté une unique fois avant les programmes “distributeur” et “ecran”.

4. Synchronisation de processus[4]

Nous vous demandons d'écrire :

- Un programme "client" qui prend en paramètre une chaîne de 10 caractères minuscules. Il écrit ces 10 caractères dans une mémoire partagée.
- Un programme "serveur" qui traduit le texte de la mémoire partagée en majuscules. Lorsque cette traduction est terminée, le programme "client" affiche le texte modifié sur "stdin".

Vos deux programmes doivent se synchroniser à l'aide de sémaphores.

5. Modularisation

Nous vous demandons d'écrire un module relatif aux sémaphores et un module relatif à la mémoire partagée. C'est deux modules définissent un ensemble de structures et de fonctions que vous pourrez réutiliser dans le cadre de votre projet de programmation réseau.

Pratiquement, nous vous demandons d'écrire les fichiers suivants :

- un fichier entête sem.h et un fichier source sem.c qui définissent et contiennent des structures et des fonctions génériques liées aux sémaphores.
- un fichier entête shm.h et un fichier source shm.c qui définissent et contiennent des structures et des fonctions génériques liées à la mémoire partagée.

Une fois ces deux modules écrits, mettez à jour la solution d'un des exercices précédents et écrivez un fichier *makefile* pour générer les exécutables.

[4] M. Bagein, S. Frémal, S. Mahmoudi et P. Manneback, *Ibid*