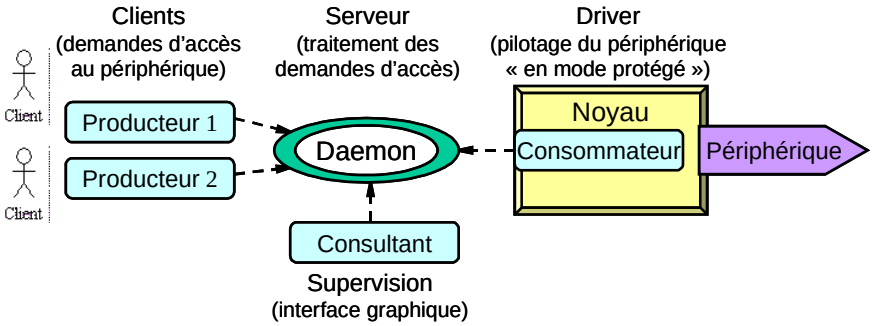
Master Ingénierie du Logiciel Libre (I2L) – 2ème Année
Module système, réseaux et sécurité
TP N°4 : Communication et synchronisation

D. DUVIVIER (duvivier@lil.univ-littoral.fr) – LIL/ULCO

Ce TP a pour but de mettre en œuvre une partie de ce qui a été vu en cours concernant la communication et synchronisation entre processus. Pour réaliser ce TP, vous vous baserez sur l'ensemble des programmes fournis (mise en œuvre des signaux, d'un segment de mémoire partagée, d'un tableau de sémaphore, de la fonction fork()…).

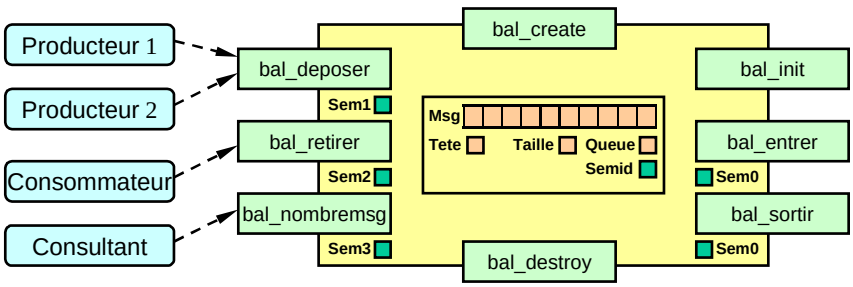
Ce TP fait partie d'un projet visant à réaliser une « chaîne de traitements » complète depuis plusieurs processus clients qui s'adressent à un processus daemon jouant le rôle d'un serveur de requêtes. Ce daemon est en interaction avec un processus du noyau chargé à son tour de transformer chaque requête en un ensemble de commandes élémentaires à destination d'un périphérique via un « driver » adapté :



Il s'agit de réaliser un pseudo-moniteur « bal » chargé de gérer un canal de communication à la manière d'une « boîte aux lettres » entre plusieurs producteurs (chargés de déposer des messages) et un consommateur (chargé de lire les messages). Afin de superviser le fonctionnement de la boîte aux lettres, un processus « consultant » relève périodiquement le nombre de messages situés dans la boîte aux lettres.

Le pseudo-moniteur « bal » est constitué d'une structure de données placée dans un segment de mémoire partagé entre les divers processus. Cette structure de données comporte une file de messages stockée dans un tableau de messages géré en « file circulaire » ainsi que diverses variables utilisées pour gérer la file de message. Parmi ces variables figure l'identificateur du tableau (semid) de quatre sémaphores utilisés pour gérer la synchronisation :

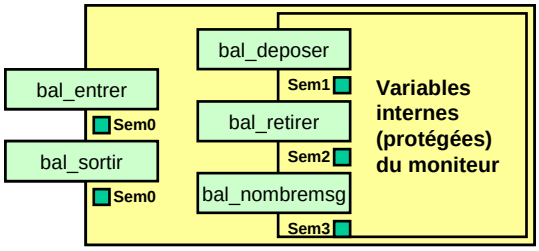
- le sémaphore N°0 gère les accès en exclusion mutuelle au pseudo-moniteur via les procédures « bal_entrer » et « bal_sortir » ;
- le sémaphore N°1 gère les accès des producteurs au pseudo-moniteur via la procédure « bal_deposer » ;
- le sémaphore N°2 gère les accès du consommateur au pseudo-moniteur via la procédure « bal_retirer » ;
- le sémaphore N°3 gère les accès du consultant au pseudo-moniteur via la procédure « bal_nombremsg ».



Les procédures bal_deposer, bal_retirer et bal_monbremsg doivent bien évidemment effectuer respectivement un appel à P(sem1), P(sem2) et P(Sem3), mais également obtenir l'autorisation d'accès exclusif au pseudo-moniteur bal par le biais du sémaphore sem0 géré par les procédures bal_entrer (qui effectue un appel à P(sem0)) et bal_sortir (qui effectue un appel à V(sem0)).

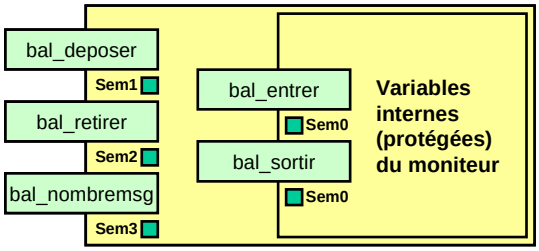
Le problème est de déterminer quand et au travers de quelle(s) procédure(s) effectuer les appels à V(sem1), V(sem2) et V(sem3) mais également dans quel ordre effectuer les différents appels (P(sem0), V(sem0), P(sem1), V(sem1), P(sem2), V(sem2), P(sem3), V(sem3)). Par exemple en ce qui concerne la procédure bal_deposer, faut-il effectuer un appel à P(sem0) avant un appel à P(sem1) ou inversement ?

Dans le premier cas, l'accès exclusif au moniteur est demandé (exclusion mutuelle entre tous les processus via un appel à P(sem0)) avant d'effectuer une demande de dépôt au travers de l'appel à P(sem1) dans la procédure bal_deposer.



Dans cette solution (solution N°1) tous les processus sont sérialisés dans la file du sémaphore 0 et gérés dans cet ordre sans possibilité de gérer des priorités (en général le consommateur est prioritaire par rapport au producteur pour éviter les débordement de la bal) et avec un risque de blocage si le processus consommateur effectue un P(sem0) alors que la bal est vide (*i.e.* aucun message à lire).

Dans le second cas (solution N°2), la demande de dépôt est effectuée au travers de l'appel à P(sem1) dans la procédure bal_deposer avant de demander l'accès exclusif au moniteur (via un appel à P(sem0).



De ce fait, le processus en cours d'accès (exclusif) au moniteur peut déterminer le nombre de processus bloqués sur chacune des files des sémaphores sem1 à sem3. Cependant les accès aux sémaphores 1 à 3 ne sont pas effectués en exclusion mutuelle (par exemple un processus peut effectuer un appel à P(sem1) alors qu'un autre processus effectue simultanément un appel à P(sem2) tandis qu'un processus consultant est dans le moniteur (i.e. il a effectué les appels à P(sem3) et P(sem0) et a été autorisé à accéder au moniteur). Les seules variables utilisables en toute sécurité par le processus situé « dans le moniteur » sont les variables internes du moniteur (i.e. même les tailles des files des sémaphores ne sont pas protégées en exclusion mutuelle sur l'ensemble des processus).

Dans la solution N°2 qui a été implémentée dans le programme qui vous a été remis, nous avons initialisé les sémaphores aux valeurs suivantes :

- Sem0 : initialisé à 1 (mutex de tous les processus sur le moniteur bal)
- Sem1 : initialisé à 10 (si l'on suppose que la file peut contenir 10 messages)
- Sem2 : initialisé à 0 (impossible de retirer un message car la file est vide initialement)
- Sem3 : initialisé à 1 (un seul consultant autorisé, dès le début, à superviser périodiquement le nombre de messages situés dans la file)

Le processus consultant peut accéder à la bal sans contrainte particulière (il est toujours possible d'obtenir le nombre de messages en bal) si ce n'est l'exclusion mutuelle lors de l'entrée dans le moniteur (via sem0).

Le processus consommateur peut accéder à la bal s'il y a au moins un message dans la file (bal non vide).

Un processus producteur peut accéder à la bal s'il y a moins de 10 messages dans la file (bal non pleine).

Afin d'éviter le problème posé par la solution N°1 (voir note de bas de page[1]) nous paramétrons les sémaphores 1 et 2 de manière à ne débloquer un producteur que lorsque la bal n'est pas pleine et à ne débloquer un consommateur que

[1] Dans la solution N°1, le fait de ne pas pouvoir gérer l'ordre de traitement (ou la priorité) des producteurs par rapport aux consommateurs fait qu'un blocage intervient lorsque la bal est vide et qu'un processus consommateur est situé avant un processus producteur dans la file du sémaphore 0, ou lorsque la bal est pleine et qu'un processus producteur est situé avant un processus consommateur dans la file du sémaphore0.

lorsque la bal n'est pas vide. Ces « conditions de déblocage » des sémaphores 1 et 2 déterminent l'instant où un processus consommateur ou producteur est débloqué de la file du sémaphore 1 ou 2 pour aller se bloquer dans la file du sémaphore 0 (sérialisation des accès au moniteur). Nous nous retrouvons dans le cas de la solution 1, cependant l'ordre de passage entre les producteurs (sémaphore 1) et des consommateurs (sémaphore 2) est géré par ces « conditions de déblocage ». Dans ces conditions de déblocages, nous choisissons de donner la priorité au consommateur par rapport au producteur (pour éviter de faire déborder la bal). Ces conditions s'écrivent comme suit si nous supposons qu'il existe un seul processus consommateur :

1) **si (un consommateur est bloqué par sem1) et (bal non vide) alors**
   (autoriser le consommateur à demander l'entrée en moniteur via sem0) ;
2) **si (au moins un producteur est bloqué par sem2) et (taille(bal) + nombre de producteurs bloqués par sem1 < 10) alors**
   (autoriser un producteur à demander l'entrée en moniteur via sem0) ;
3) **si (un consultant est bloqué par sem3) alors**
   (autoriser un consommateur à demander l'entrée en moniteur via sem0).

Question : Est-ce suffisant, efficace et « correct » pour ne provoquer ni famine (livelock), ni blocage (deadlock), ni erreur (bad luck !) ?