ENSEA
Ecole Nationale
Supérieure
de l'Electronique
et de ses Applications

3e année – IS – Systèmes et Réseaux

TP 3 – Communication client-serveur par segment mémoire partagé protégée par sémaphores

C. BARÈS

Objectifs

Le but de ce TP est d'utiliser les mécanisme IPC pour synchroniser un ensemble de programme *producer/consumer*.

Dans le cadre du TP, seul le processus client est à réaliser.

1 – Principe

Le processus client fournit au serveur des tableaux de valeurs aléatoires dont il calcule la moyenne. Le processus client va faire un certain nombre de requêtes auprès du serveur et entrer en concurrence avec d'autres processus pour accéder à la ressource commune, ici, le segment mémoire.

Pour gérer les conflits d'accès, 3 sémaphores sont mis en place :

- `seg_dispo` : protège l'accès au segment mémoire. L'acquisition de ce sémaphore indique que l'on peut utiliser le segment.
- `seg_init` : L'acquisition de ce sémaphore par le client indique au serveur que le segment est initialisé.
- `res_ok` : L'acquisition de ce sémaphore par le serveur indique au client que le résultat est prêt.

Afin de tester la cohérence du fonctionnement, le client calculera sa propre moyenne pour la comparer avec celle fournie par le serveur. De plus, le client transmettra son numéro de pid et un numéro de requête lors de chaque demande et vérifiera lorsqu'il récupérera le résultat qu'il correspond bien à la requête envoyée.

2 – Dialogue

Le dialogue entre les 2 processus est le suivant :

client		serveur	
0	récupère les identifiant du segment, des sémaphores et initialise le générateur.	0	Crée le segment mémoire. Crée et initialise les sémaphores.
1	demande à acquérir `seg_dispo`	1	attend que `seg_init` passe à 0
2	initialise segment et acquiert `seg_init`	2	calcule résultat et acquiert `res_ok`
3	attend que `res_ok` passe à 0	3	essaie d'acquérir `seg_init` (indique résultat lu)
4	lit résultat et libère `seg_init`	4	libère `res_ok` puis `seg_init`
5	attendre la libération de `res_ok`	5	boucle sur l'étape 1
6	libère `seg_dispo`		
7	affiche résultats		
8	boucle sur l'étape 1		

Un certain nombres de procédures réunies dans la librairie `libseg.a` sont disponibles pour réaliser le programme :

`init_rand()` : initialise le générateur
`long getrand()` : retourne un entier long signé

Les routines suivantes bloquent le processus en attendant d'aboutir. semid désigne l'identifiant de l'ensemble de sémaphores :

`wait_sem(semid, sem)` : attend que le sémaphore sem passe à 0
`acq_sem(semid,sem)` : tente d'acquérir le sémaphore sem
`lib_sem(semid,sem)` : libère le sémaphore sem

La structure du segment sera définie ainsi :

```
struct shmseg
{
      int pid;            /* Numero de pid du client*/
      int req;            /* numero de requete */
      long tab[maxval];   /* Tableau de valeurs */
      long result;        /* resultat */
}
```

Cette structure est définie sous le type SEGMENT dans le fichier segdef.h. Ce fichier réunit tous les includes nécessaires ainsi que la définition des constantes suivantes :

`cle`	(`key_t`)3	clé d'accès commune au segment et aux sémaphores
`seg_dispo`	0	sémaphore accès segment
`seg_init`	1	sémaphore segment initialisé
`res_ok`	2	sémaphore résultat prêt
`maxval`	100	nombre de valeurs à calculer
`segsize`		taille du type SEGMENT

L'application sera conçue à partir de 4 fichiers :

- `libseg.a` : bibliothèque pré-compilée
- `segserv` : le programme serveur pré-compilé
- `segdef.h` : fichier de déclaration à inclure dans votre code

- `xxxxx.c` : votre programme client

Le fichier client devra déclarer une structure associée à la manipulation des sémaphores : `struct sembuf sop;`

L'organisation du programme sera la suivante :

1. Initialisations (faire une fonction séparée)
    - récupérer les identifiants segment et sémaphore avec détection d'erreurs
    - attacher le segment avec détection d'erreurs
    - initialiser le générateur
2. Boucle contrôlée par le compteur de requête
    - demander à acquérir `seg_dispo`
    - initialiser le segment et calculer le résultat local
    - demander à acquérir `seg_init`
    - attendre `res_ok`
    - libérer résultat et `seg_init`
    - libérer `seg_dispo`
    - afficher résultats comparés client et serveur
3. détacher segment mémoire et exit

2.1 Travail à effectuer

1. Créer le makefile de votre application ;
2. Réaliser le client selon le modèle ci-dessus ;
3. écrire vos propres routines de manipulation des sémaphores et remplacer la bibliothèque libseg.a par la votre.