Communication par segment de mémoire partagée [IPC System V]

EXERCICE 1 :

Implémentez les fonctions suivantes permettant la gestion des segments de mémoire partagée (voir fichiers fournis `segment_memoire.h` et `segment_memoire.c`) :

↘ `int cree_segment(int taille, char* nom, int cle)` ; crée un segment de mémoire (en utilisant la fonction `shmget`)

- `taille` : est la taille du segment de mémoire à créer ;
- `nom` : est le nom du fichier associé ;
- `cle` : associée au nom pour la construction d'une clé identifiant le segment de mémoire (en utilisant la fonction `ftok`) ;
- valeur de retour : l'identificateur du segment crée.

↘ `int afficher_info_segment(int shmid)` ; affiche toutes les informations concernant un segment de mémoire (en utilisant la fonction `shmctl`)

- `shmid` : l'identificateur du segment.
- valeur de retour : -1 en cas d'erreur.

↘ `int detruire_segment(int shmid)` ; détruit un segment de mémoire (en utilisant la fonction `shmctl`)

- `shmid` : l'identificateur du segment.
- valeur de retour : -1 en cas d'erreur.

  Exemple d'utilisation : voir `exo1.c` (annexe 1).

EXERCICE 2 :

Utilisez les fonctions implémentées en *exercice 1* pour développer le programme `exo2.c` (annexe 1) : le processus père écrit un message dans un segment de mémoire partagée. Ensuite il crée un processus fils qui affiche le contenu du segment (attaché en lecture). Le processus père attend la fin du fils avant de détacher le segment et de le détruire.

Annexe 1

/* segment_memoire.h */

```
/*****************************************************************
 * OS302 - shm - Exercice 1                                      *
 * Utilisation des segments de memoire partagee                  *
 *****************************************************************/
#ifndef SEGMENT_MEM_H
#define SEGMENT_MEM_H

#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>

/* Cree un segment de memoire
 * taille : la taille du segment memoire a creer
 * nom    : le nom du fichier associe
 * cle    : associe au nom pour la construction d'une cle identifiant
 *          le segment memoire (on utilisant la fonction ftok)
 * retourne : l'identificateur du segment, ou -1 en cas d'erreur */
int cree_segment(int taille, char* nom, int cle);

/* Afficher toutes les informations concernant un segment de memoire
 * shmid : l'identificateur du segment
 * retourn -1 en cas d'erreur */
int afficher_info_segment(int shmid);


/* Detruie un segment de mémoire
 * shmid : l'identificateur du segment
 * retourn -1 en cas d'erreur */
int detruire_segment(int shmid);
```

/* exo1.c */

```
/*****************************************************************
 * OS302 - shm - Exercice 1                                      *
 * Utilisation des segments de memoire partagee                  *
 *****************************************************************/
#include "segment_memoire.h"

int main() {
  int shmid ;
  char *nom = (char *) malloc(100*sizeof(char)) ;
  nom = "exo1.c" ;
  shmid = cree_segment(100,nom,1) ;
  afficher_info_segment(shmid) ;
```

```
  detruire_segment(shmid) ;
  return 0 ;
}
```

/* exo2.c */

```
/*****************************************************************
 * OS302 - shm - Exercice 2                                      *
 * Utilisation des segments de memoire partagee                  *
 *****************************************************************/
#include <string.h>
#include <unistd.h>
#include "segment_memoire.h"

int main() {

  int pid;      // le PID du processus
  char *mem;    // pointeur vers le segment memoire
  int shmid;    // l'identificateur du segment memoire
  char *nom = (char *) malloc(15*sizeof(char));
  nom = "exo2.c";

  // a completer : creation du segment de memoire partagee
  // ...

  // creation du processus fils
  pid = fork();
  if (pid == -1) {
    perror("impossible de creer un processu fils!");
    exit(-1);
  }
  else if (pid == 0) {
    sleep(2);
    // je suis le fils!
    // a completer : s'attacher au segment et affichage de son contenu
    // ...
  }
  else {
    // je suis le pere!
    // a completer : attachement et ecriture sur le segment de memoire partagee
    // a completer : attendre la fin du fils + detacher le segment et le detruire
    // ...
  }

  return 0 ;
}
```

Annexe 2

Pour gérer des zones de mémoire communes à plusieurs processus, Unix propose les primitives suivantes :

- `shmget`, (création d'une zone de mémoire partagée)
- `shmctl`, (obtenir des informations sur une zone)
- `shmat`, (attacher une zone à un processus via un pointeur)
- `shmdt`, (rendre une zone)

Les structures de données nécessaires à la gestion des zones de mémoires partagées sont répertoriées dans :

/usr/include/sys/shm.h

/usr/include/sys/ipc.h

/usr/include/sys/types.h

Nous allons décrire les quatre fonctions : shmget, shmctl, shmat et shmdt.

.1.1 shmget

```
int shmget(int mem_clef, int taille, int drapeau)
```

shmget renvoie un descripteur local identifiant la zone mémoire associée à la clef mem_clef. Si le bit IPC_CREAT est mis à 1 dans la variable drapeau ou si la clé vaut IPC_PRIVATE, alors shmget créé et initialise une zone de mémoire partageable, c'est-à-dire un objet de type shmid_ds (décrit dans /usr/include/sys/shm.h). Sinon, il positionne simplement l'utilisateur sur cette zone de clef mem_clef. Si IPC_PRIVATE est mis à 1, alors la zone n'est vue que par le créateur et ses descendants.

.1.2 shmat

```
char *shmat(int shmid, char *adresse, int drapeau)
```

shmat renvoie un pointeur sur la zone identifiée par shmid (ce dernier est l'entier renvoyé par shmget). adresse est un pointeur sur caractère qui indique à quelle adresse mémoire (virtuelle) l'utilisateur veut implanter cette zone, NULL veut dire que l'on charge le système de ce travail (option fortement conseillée !)

.1.3 shmdt

```
int shmdt(char *adresse)
```

shmdt signifie que le processus n'utilise plus la zone mémoire commune. Quand le nombre d'utilisateurs de cette zone tombe à 0, elle est restituée au système.

.1.4 shmctl

```
int shmctl(int shmid, int commande, struct shmid_ds *tab)
```

shmctl envoie la commande commande sur la zone shmid. Suivant la commande envoyée, cette fonction renvoie des informations dans tab, ou modifie les caractéristiques de la zone.